

DATABASE DESIGN DESCRIPTION
FOR THE
TACTICAL ENVIRONMENTAL SUPPORT SYSTEM/
NEXT CENTURY [TESS(3)/(NC)]
METEOROLOGICAL/OCEANOGRAPHIC (METOC) DATABASE

30 SEPTEMBER 1997
PRELIMINARY

Prepared for:
Space and Naval Warfare Systems Command
METOC Systems Program Office
(SPAWAR PMW-185)
and
Naval Research Laboratory
Monterey, CA

Prepared by:
Integrated Performance Decisions, Inc.
Monterey, CA

PRELIMINARY

TABLE OF CONTENTS

1	SCOPE	1-1
1.1	Identification.....	1-1
1.2	Database Overview	1-1
1.3	Document Overview	1-1
2	REFERENCED DOCUMENTS.....	2-1
2.1	Government Documents	2-1
2.1.1	Military Specifications.....	2-1
2.1.2	Military Standards	2-1
2.1.3	Other Military Documents	2-1
3	DATABASE-WIDE DESIGN DECISIONS	3-1
3.1	Database Components	3-1
3.2	Database Organization	3-2
3.2.1	NITES I METOC Database	3-2
3.2.2	Central Site METOC Database	3-3
3.2.3	NITES II METOC Database	3-3
3.2.4	Satellite Analysis and Viewer Database	3-4
3.3	Database Management System.....	3-5
3.4	Database Access	3-6
3.4.1	Modes.....	3-6
3.4.2	Discretionary Access	3-6
3.4.3	METOC Database APIs.....	3-7
3.4.3.1	ANSI Standard SQL	3-7
3.4.3.2	Connect Operation	3-7
3.4.3.3	TEDS APIs.....	3-8
3.4.3.4	Operating System File System Services	3-8
3.5	Performance Tradeoff Analysis	3-9
4	DETAILED DESIGN OF THE DATABASE.....	4-1
4.1	Grid Field Data Segment (MDGFD) Design.....	4.1-1
4.1.1	MDGFD Conceptual Level Design	4.1-1
4.1.2	MDGFD Logical Level Design.....	4.1-2
4.1.3	MDGFD Physical Level Design.....	4.1-3
4.1.3.1	Area of Interest Table	4.1-4
4.1.3.2	Geophysical Parameters Table	4.1-5
4.1.3.3	Grid Data Detail Table.....	4.1-6
4.1.3.4	Grid Dataset Directory	4.1-7

PRELIMINARY

4.1.3.5	Lambert/Mercator Projection Table	4.1-9
4.1.3.6	Linear Conversion Table.....	4.1-10
4.1.3.7	Model Reference Table.....	4.1-11
4.1.3.8	Polar Stereographic Projection Table	4.1-12
4.1.3.9	Production Centers Table.....	4.1-13
4.1.3.10	Registrations Table	4.1-14
4.1.3.11	Spherical Projection Table	4.1-16
4.1.3.12	Units Table	4.1-17
4.2	LLT Observation Data Segment (MDLLT) Design	4.2-1
4.2.1	MDLLT Conceptual Design.....	4.2-1
4.2.2	MDLLT Logical Level Design.....	4.2-3
4.2.3	MDLLT Physical Level Design	4.2-12
4.2.3.1	Database Tables Common to All Observation Types.....	4.2-12
4.2.3.2	Physical Level Design for Surface Observations	4.2-15
4.2.3.3	Physical Level Design for Upper Air Observations	4.2-30
4.2.3.4	Physical Level Design for Ocean Observations	4.2-52
4.2.3.5	Physical Level Design for METAR, SPECI, and TAF.....	4.2-58
4.3	Textual Observations Data Segment (MDTXT) Design.....	4.3-1
4.3.1	MDTXT Conceptual Design	4.3-1
4.3.2	Physical Level Design for Textual Observations and Bulletins Storage.....	4.3-1
4.3.2.1	Text Bulletins Table	4.3-3
4.3.2.2	Textual Observations/Bulletins Dataset Directory Table	4.3-5
4.4	Image Data Segment (MDIDS) Design	4.4-1
4.4.1	MDIDS Conceptual Design	4.4-1
4.4.2	MDIDS Logical Level Design	4.4-1
4.4.3	Physical Level Design of MDIDS	4.4-2
4.4.3.1	Image Dataset Table	4.4-3
4.4.3.2	Image Dataset Directory Table.....	4.4-4
4.5	Remotely Sensed Data Segment (MDRSD).....	4.5-1
4.5.1	MDRSD Conceptual Level Design	4.5-1
4.5.2	MDRSD Logical Level Design	4.5-1
4.5.3	MDRSD Physical Level Design.....	4.5-3
4.5.3.1	Channels Table	4.5-4
4.5.3.2	Ephemeris Table	4.5-5
4.5.3.3	Remotely Sensed Dataset Table	4.5-6
4.5.3.4	Remotely Sensed Dataset Directory Table	4.5-8
4.5.3.5	Satellite Table.....	4.5-9
4.5.3.6	Satellite Sensors Table.....	4.5-10
4.5.3.7	Sensor Table.....	4.5-11

PRELIMINARY

4.6	Database Utilities	4.6-1
4.6.1	Create Table Utility Tables	4.6-1
4.6.1.1	Canned SQL Table	4.6-2
4.6.1.2	Data Type to SQL Table	4.6-3
4.6.1.3	Observation Subtypes Table.....	4.6-4
4.6.1.4	Observation Types Table	4.6-5
5	DETAILED DESIGN OF SOFTWARE UNITS USED FOR DATABASE ACCESS OR MANIPULATION	5-1
5.1	Interfaces for 2-D Gridded Data	5-1
5.1.1	Retrieving 2-D Gridded Data From the TEDS Database.....	5-1
5.1.2	Storing 2-D Gridded Data in the TEDS Database.....	5-7
5.1.3	Opening a 2-D Grid Dataset.....	5-8
5.1.4	Adding 2-D Gridded Data to a Dataset.....	5-9
5.1.5	Retrieving a Catalog Listing of 2-D Gridded Data	5-10
5.1.6	Retrieving 2-D Gridded Data By Reference ID	5-11
5.2	Interfaces for Bathythermograph Report (BT) Data	5-12
5.2.1	Getting a Catalog of BT Data From the Database.....	5-12
5.2.2	Retrieving a Specific BT Record From the Database.....	5-13
5.2.3	Retrieving Multiple BT Records From the Database	5-14
5.2.4	Deleting a BT Record From the Database	5-15
5.2.5	Adding a BT Record to the Database	5-16
5.2.6	Adding a BT Record to a Dataset.....	5-17
5.2.7	Updating a BT Record.....	5-18
5.3	Interfaces for Ocean Profile (OP) Data	5-19
5.3.1	Getting a Catalog of OP Data From the Database.....	5-19
5.3.2	Retrieving a Specific OP Record From the Database.....	5-20
5.3.3	Retrieving Multiple OP Records From the Database	5-21
5.3.4	Deleting an OP Record From the Database	5-22
5.3.5	Adding an OP Record to the Database	5-23
5.3.6	Adding an OP Record to a Dataset.....	5-24
5.3.7	Updating an OP Record	5-25
5.4	Interfaces For Upper Air Report (UA) Data	5-26
5.4.1	Getting a Catalog of UA Data From the Database	5-26
5.4.2	Retrieving a Specific UA Record From the Database	5-27
5.4.3	Retrieving Multiple UA Records From the Database.....	5-28
5.4.4	Deleting an UA Record From the Database.....	5-29
5.4.5	Adding a UA Record to the Database.....	5-30
5.4.6	Adding a UA Record to a Dataset	5-31

PRELIMINARY

5.4.7	Updating a UA Record	5-32
5.5	Interfaces For Upper Air Profile (UP) Data.....	5-33
5.5.1	Getting a Catalog of UP Data From the Database.....	5-33
5.5.2	Retrieving a Specific UP Record From the Database.....	5-34
5.5.3	Retrieving Multiple UP Records From the Database	5-35
5.5.4	Deleting a UP Record From the Database	5-36
5.5.5	Adding an UP Record to the Database	5-37
5.5.6	Adding a UP Record to a Dataset.....	5-38
5.5.7	Updating a UP Record.....	5-39
5.6	Interfaces For Synoptic Report (SN) Data	5-40
5.6.1	Getting a Catalog of SN Data From the Database.....	5-40
5.6.2	Retrieving a Specific SN Record From the Database.....	5-41
5.6.3	Retrieving Multiple SN Records From the Database	5-42
5.6.4	Deleting a SN Record From the Database	5-43
5.6.5	Adding a SN Record to the Database	5-44
5.6.6	Adding a SN Record to a Dataset.....	5-45
5.6.7	Updating an SN Record	5-46
5.7	Interfaces For Vertical Sounding (VS) Data	5-47
5.7.1	Getting a Catalog of VS Data From the Database.....	5-47
5.7.2	Retrieving a Specific VS Record From the Database.....	5-48
5.7.3	Retrieving Multiple VS Records From the Database	5-49
5.7.4	Deleting a VS Record From the Database	5-50
5.7.5	Adding a VS Record to the Database	5-51
5.7.6	Adding a VS Record to a Dataset.....	5-52
5.7.7	Updating a VS Record	5-53
5.8	Interfaces for AIRCRAFT (AC) Data	5-54
5.8.1	Getting a Catalog of AC Data From the Database	5-54
5.8.2	Retrieving a Specific AC Record From the Database	5-55
5.8.3	Retrieving Multiple AC Records From the Database	5-56
5.8.4	Deleting an AC Record From the Database	5-57
5.8.5	Adding an AC Record to the Database	5-58
5.8.6	Adding an AC Record to a Dataset	5-59
5.8.7	Updating an AC Record.....	5-60
5.9	Interfaces For SSM/I Observation Data	5-61
5.9.1	Getting a Catalog of SSM/I Data from the Database	5-61
5.9.2	Retrieving a Specific SSM/I Record From the Database.....	5-62
5.9.3	Retrieving Multiple SSM/I Records From the Database	5-63
5.9.4	Deleting an SSM/I Record From the Database	5-64
5.9.5	Adding an SSM/I Record to the Database	5-65

PRELIMINARY

5.9.6	Adding an SSM/I Record to a Dataset.....	5-66
5.9.7	Updating an SSM/I Record.....	5-67
5.10	Interfaces for Textual Observations and Bulletins.....	5-68
5.10.1	Getting a Catalog of TXT Data from the Database	5-68
5.10.2	Retrieving a Specific TXT Record From the Database	5-69
5.10.3	Retrieving Multiple TXT Records From the Database.....	5-70
5.10.4	Deleting a TXT Record From the Database.....	5-71
5.10.5	Adding a TXT Record to the Database.....	5-72
5.10.6	Adding a TXT Record to a Dataset	5-73
5.10.7	Updating a TXT Record	5-74
5.11	Interfaces for TRACK WIND Data	5-75
5.11.1	Getting a Catalog of TRACKWIND Data from the Database	5-75
5.11.2	Retrieving a Specific TRACKWIND Record From the Database	5-76
5.11.3	Retrieving Multiple TRACKWIND Records From the Database.....	5-77
5.11.4	Deleting a TRACKWIND Record From the Database.....	5-78
5.11.5	TRACKWIND Ingest Interfaces	5-79
5.11.6	Updating a TRACKWIND Record	5-81
5.12	Interfaces For scatterOMETRY Data	5-82
5.12.1	Getting a Catalog of SCATTER Data from the Database	5-82
5.12.2	Retrieving a Specific SCATTER Record From the Database	5-83
5.12.3	Retrieving Multiple Scatterometry Records From the Database.....	5-84
5.12.4	Deleting a Scatterometry Record From the Database.....	5-85
5.12.5	Ingesting Scatterometry Data.....	5-86
5.12.6	Updating a Scatterometry Record	5-88
5.13	Interfaces for Image Data.....	5-89
5.13.1	Retrieving TIFF Imagery	5-89
5.13.2	Storing TIFF Imagery in the TEDS Database.....	5-90
6	REQUIREMENTS TRACEABILITY	6-1
7	NOTES.....	7-1
7.1	Glossary of acronyms	7-1

PRELIMINARY

LIST OF FIGURES

3.1-1	METOC Database Organization – Conceptual View	3-1
4.1-1	Logical Level Design of the MDGFD	4.1-2
4.1-2	Physical Level Design of the MDGFD.....	4.1-3
4.2-1	Ocean Observation Data Logical Model.....	4.2-4
4.2-2	Surface Observation Data Logical Model.....	4.2-5
4.2-3	Radar Observation Data Logical Model	4.2-6
4.2-4	Upper Air Observation Data Logical Model (Part 1 of 2).....	4.2-7
4.2-5	Remotely Sensed Data Logical Model	4.2-9
4.2-6	Logical Model for Image Data.....	4.2-10
4.2-7	Logical Model for LLT Data Location Information.....	4.2-11
4.2-8	Physical Level Design for Surface Observations	4.2-15
4.2-9	Physical Level Design for Storage of Upper Air Observations	4.2-30
4.2-10	Physical Level Design of Storage for Ocean Observations	4.2-52
4.2-11	Physical Level Design for Storage of METAR, SPECI, and TAF Reports	4.2-58
4.3-1	Physical Level Design for Textual Observations and Bulletins Storage.....	4.3-2
4.4-1	Logical Level Design for MDIDS	4.4-1
4.4-2	Physical Level Design of MDIDS	4.4-2
4.5-1	MDRSD Logical Level Design	4.5-2
4.5-2	MDRSD Physical Level Design.....	4.5-3
4.6-1	Physical Design of Create Table Tables	4.6-1

PRELIMINARY

LIST OF TABLES

4.1-1	AOIs Structure.....	4.1-4
4.1-2	GeoPhysicalParms Structure.....	4.1-5
4.1-3	gridData Structure	4.1-6
4.1-4	Grid Data Set Directory Structure	4.1-7
4.1-5	LambertMercator Structure.....	4.1-9
4.1-6	LinearConversions Structure.....	4.1-10
4.1-7	Models Structure	4.1-11
4.1-8	PolarStereographic Structure	4.1-12
4.1-9	ProductionCenters Structure	4.1-13
4.1-10	Registrations Structure.....	4.1-14
4.1-11	Spherical Structure	4.1-16
4.1-12	Units Structure.....	4.1-17
4.2-1	obAOIs Structure.....	4.2-12
4.2-2	bCollectionAreas Structure	4.2-13
4.2-3	ObDataSetDir Structure	4.2-14
4.2-4	cloudData Structure	4.2-16
4.2-5	conditions Structure	4.2-17
4.2-6	cycloneConditions Structure	4.2-18
4.2-7	landStations Structure	4.2-20
4.2-8	oceanData Structure.....	4.2-21
4.2-9	radarOb Structure	4.2-23
4.2-10	ships Structure	4.2-25
4.2-11	surfaceOB Structure	4.2-26
4.2-12	synopticOb Structure	4.2-28
4.2-13	AircraftReports Structure.....	4.2-31
4.2-14	cloudInfo Structure	4.2-33
4.2-15	convCond Structure	4.2-34
4.2-16	evaporationHt Structure	4.2-35
4.2-17	rocketSounding Structure.....	4.2-36
4.2-18	shipReport Structure	4.2-38
4.2-19	stationConditions Structure.....	4.2-39

PRELIMINARY

4.2-20	stationStructure.....	4.2-41
4.2-21	UAProfile Structure.....	4.2-42
4.2-22	uaProfileSounding Structure	4.2-44
4.2-23	UATemperature Structure.....	4.2-46
4.2-24	UATempSounding Structure.....	4.2-47
4.2-25	UAWindSounding Structure	4.2-49
4.2-26	UpperAirRep Structure	4.2-50
4.2-27	bathySounding Structure.....	4.2-53
4.2-28	buoy Structure	4.2-54
4.2-29	buoySounding Structure	4.2-55
4.2-30	oceanOBData Structure	4.2-56
4.2-31	aerodromeReports Structure	4.2-59
4.2-32	cloudData Structure	4.2-61
4.2-33	forecasts Structure	4.2-62
4.2-34	METARSPECIReport Structure	4.2-63
4.2-35	runwayConditions Structure	4.2-64
4.2-36	TAFReport Structure	4.2-65
4.3-1	textBulletins Structure	4.3-3
4.3-2	textDataSetDir Structure	4.3-5
4.4-1	imageDataset Structure	4.4-3
4.4-2	imageDatasetDir Structure	4.4-4
4.5-1	Channels Structure.....	4.5-4
4.5-2	Ephem Structure	4.5-5
4.5-3	RS_Dataset Structure.....	4.5-6
4.5-4	RS_Datasets Structure	4.5-8
4.5-5	Satellite Structure	4.5-9
4.5-6	SatSensors Structure	4.5-10
4.5-7	Structure	4.5-11
4.6-1	cannedSQL Structure.....	4.6-2
4.6-2	dataTypeToSQL Structure	4.6-3
4.6-3	obSubTypes Structure.....	4.6-4
4.6-4	obTypes Structure.....	4.6-5

PRELIMINARY

1 SCOPE

1.1 Identification

1.2 Database Overview

[This paragraph shall briefly state the purpose of the database to which this document applies. It shall describe the general nature of the database; summarize the history of its development, use, and maintenance; identify the project sponsor, acquirer, user, developer, and support agencies; identify current and planned operating sites; and list other relevant documents.]

1.3 Document Overview

The remainder of this document is organized as follows:

Section 2 Lists referenced documents.

Section 3 Discusses decisions about the database's behavioral design and other decisions affecting further design of the database.

Section 4 Describes the detailed design of the database.

Section 5 Describes the detailed design of software units used for database access or manipulation.

Section 6 Provides requirements traceability to applicable requirements documents.

Section 7 Provides notes, including a glossary of acronyms.

PRELIMINARY

2 REFERENCED DOCUMENTS

2.1 Government Documents

2.1.1 Military Specifications

Space and Naval Warfare Systems Command, Environmental Systems Program Office (SPAWAR PMW-185), Washington, DC

Unnumbered	Software Requirements Specification (SRS) for the
30 September	Tactical Environmental Support System/Next Century
	[TESS(3)/NC] Meteorological/Oceanographic (METOC) Database

2.1.2 Military Standards

MIL-STD-498	Software Development and Documentation
5 December 1995	

2.1.3 Other Military Documents

DI-IPSC-81433	Data Item Description (DID) for the Software
5 December 1995	Requirements Specification (SRS), per MIL-STD-498

Space and Naval Warfare Systems Command, Environmental Systems Program Office (SPAWAR PMW-185), Washington, DC

Unnumbered	Tactical Environmental Data System (TEDS) Release
10 June 1997	3.5 Model Data Applications Program Interface (API)
	User's Guide

Unnumbered	Tactical Environmental Data System (TEDS) Release
20 June 1997	3.5 Observation/Profile Data Applications Program

PRELIMINARY

Interface (API) User's Guide

Unnumbered

20 June 1997

Tactical Environmental Data System (TEDS) Release
3.5 Image Data Applications Program Interface (API)
User's Guide

Unnumbered

01 August 1997

Tactical Environmental Data System (TEDS) Release
3.5 Remotely Sensed Data Applications Program
Interface (API) User's Guide

PRELIMINARY

3

DATABASE-WIDE DESIGN DECISIONS

3.1

Database Components

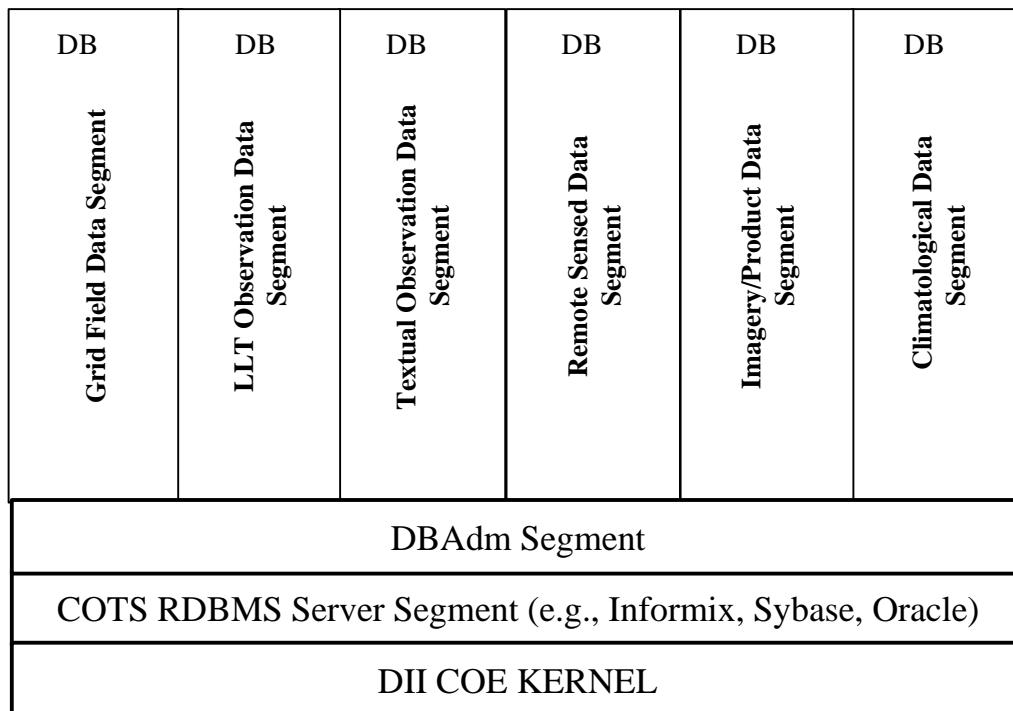


Figure 3.1-1. METOC Database Organization – Conceptual View

PRELIMINARY

3.2 Database Organization

The TESS(3)/NC Concept of Operations and system architecture require that the METOC Database be distributed both in terms of application access to METOC data and products and in terms of physical location of the data repositories. The organizational structure of the database is influenced by these requirements and the components of this distributed database are described in the following paragraphs.

3.2.1 NITES I METOC Database

The NITES I METOC Database serves as the primary source of METOC data and products for NITES I and II applications. METOC data and products distributed from the Central Sites via the JMV/HPSL applications, local external interface decode/filter processing, and local application processing (e.g., COAMPS) populate the database with grid field, observation, imagery, and climatological METOC data. The NITES I METOC Database is located at shipboard, regional centers, and detachment level METOC sites.

In accordance with DII COE and SHADE database concepts, the METOC Database is composed of six DII COE Level 5 compliant *shared database* segments. These segments names and prefixes are:

1. Grid Field Data Segment (MDGFD)
2. LLT Observation Data Segment (MDLOD)
3. Textual Observation Data Segment (MDTOD)
4. Remotely Sensed Data Segment (MDRSD)
5. Imagery Data Segment (MDIDS)
6. Climatological Data Segment (MDMCS).

The database segments are data type specific and independently installable. Each segment is designed to support multiple applications and can be installed together to support applications requiring METOC data of different types. Applications in NITES I and II systems access the database through published application

PRELIMINARY

programming interfaces (APIs). Each of the database segments will be implemented as a shared database segment as defined by the DII COE. The decomposition of the METOC Database into these segments is based on the following criteria:

- Data objects conveniently managed as a unit.
- Data objects needed together to support a functional area.
- Common sources or providers of data.
- Data interdependencies.
- Frequency of update.

Each segment provides the software scripts required to create the database schema, allocate storage, seed data, and extend DBA services. The database segment development methodology and structure is defined in Version 3.0 of the DII COE I&RTS. The METOC Database organization is conceptually represented in Figure 3.1-1.

3.2.2 Central Site METOC Database

A version of the NITES I METOC Database will reside at both FNMOC and NAVOCEANO central sites and serve as the primary data distribution points for central site generated METOC data and products. The Central Site METOC Database will support requests from and distribute data to shipboard, regional, and detachment level METOC sites. The distribution mechanism is implemented above the COTS DBMS layer in application software (i.e. JMV/HPSL) applications). The central site database is architecturally identical to the NITES I METOC Database but will be scaled, in terms of storage space and processing capability, to meet the data distribution requirements levied by the TESS(3)/NC Concept of Operations.

3.2.3 NITES II METOC Database

Applications in the NITES II system will use the NITES I METOC Database as its primary source of environmental data. Distribution of data from the NITES I METOC Database to NITES II applications will rely on the client/server

PRELIMINARY

capabilities of the COTS DBMS product and the METOC Database APIs identified in section 3.4. For those NITES II systems not co-located with a NITES I system or NITES II systems with limited or no access to the METOC LAN, existing JMCOMS capabilities will be used.

3.2.4 Satellite Analysis and Viewer Database

The AN/SMQ-11 Upgrade is an external system that ingest, processes, and displays satellite imagery and sensor data from polar orbiting and geostationary satellites. The system has implemented a physically and logically separate database to store satellite pass data, sensor data, and imagery product data generated by the system. Through the **TBD** interface, the METOC Database receives and maintains descriptive and referential data about the data and products stored in the AN/SMQ-11 Upgrade system and makes that data available to NITES I and II client applications through the MDRSD and MDIDS database segments.

PRELIMINARY

3.3 Database Management System

The METOC Database relies on a DII COE approved COT RDBMS to manage the storage and manipulation of the logically related data in the database. The specific DII COE RDBMS used by the METOC Database is Informix Version 7.2 or above. The METOC Database minimizes the dependency on vendor specific or extended features of the COTS DBMS to support change out and portability of the database to new or upgraded versions of the DBMS.

PRELIMINARY

3.4 Database Access

The METOC Database is a dynamic database populated with perishable environmental data ingested, updated, and deleted on a regular and real time basis. External interfaces, external systems, and local applications provide METOC data and products to the database. The management of the data into and out of the database is performed by the COTS DBMS and NITES I data management applications. The data management applications rely on database operations to store, update, retrieve, and delete the data. Access to the database and these operations is implemented according to mode, discretionary access mechanisms, and portable distributed METOC APIs.

3.4.1 Modes

NITES I and II applications can establish a connection to the METOC Database and perform *granted* database operations when the METOC Database is in On-line Mode only. Privileged system administration activities by the DBA are permitted when the database is in On-line or Maintenance Mode.

3.4.2 Discretionary Access

The METOC Database, by DII COE definition, will be a *public* database and generally available to all users of the system. Database client applications that use METOC Data to generate products will be provided with read (select) access to the required database segments. Data management applications or user applications that store, update, or delete data in the database will be granted those privileges on an as-required basis per agreement between the application developer and the TESS(3)/NC Chief Engineer. The *grants* and *roles* functionality provided by the COTS DBMS will be the implementation mechanism used to define application level discretionary access to the database.

PRELIMINARY

3.4.3 METOC Database APIs

The METOC Database supports application access to data through a set of layered APIs. The APIs will be public and consist of:

1. ANSI standard Structured Query Language (SQL)
2. Tactical Environmental Data System (TEDS) APIs
3. Operating system File System Services.

These APIs provide access to the database for storage, retrieval, maintenance, and distribution of METOC data and products. The APIs are used by NITES I data management applications, NITES I and II user interface applications, and non-TESS(3)/NC systems requiring METOC data. The APIs rely on the COTS DBMS client/server capabilities to facilitate local and distributed access to the data. The APIs are portable to the DII COE HP-UX, Sun Solaris, and Windows/NT platforms.

3.4.3.1 *ANSI Standard SQL*

Direct access to the database is managed by the COTS DBMS through ANSI standard SQL-92 (ANSI X3.135-1992). The COTS DBMS provides support for ANSI SQL statements that implement basic database operations through the SQL calling mechanism. The basic operations are:

1. Logically connecting to the database (connect)
2. Storing data in the database (insert)
3. Updating data in the database (update)
4. Retrieving data from the database (select)
5. Deleting data from the database (delete).

The following paragraphs discuss the behavior of the database in general terms.

3.4.3.2 *Connect Operation*

User application programs must first establish a logical connection to the METOC Database before any subsequent data access operations can be performed.

PRELIMINARY

Because the METOC Database is a federation of up to six separate databases, user applications must establish a separate connection to each database used during the session. To establish the logical connection, the user application executes a *connect* statement with the appropriate database name input argument. Upon completion of the operation, the user application will interrogate the SQLSTATE status variable updated by the database server. If the connect is successful, transaction-oriented data access operations can proceed. At the conclusion of the transaction(s), the user application should terminate the connection to the METOC Database.

3.4.3.2.1 Data Transaction Operations

The METOC Database will behave in similar fashion for the transaction-oriented data manipulation operations of *insert*, *update*, *select*, and *delete*. Once connected to a database, user applications with the appropriate grants can prepare input arguments, execute the required SQL statements, and block (i.e. wait on) the SQL call. Upon completion of the operation, the user application will interrogate the SQLSTATE status variable. If the operation is successful, data access processing can continue. Otherwise, error processing will proceed.

3.4.3.3 TEDS APIs

The TEDS APIs provide a programming convenience layer for application developers and abstract the data management and implementation details of the METOC Database away from the developer. The TEDS APIs are a set of application callable C Language routines implemented as static and shared libraries. Section 5 discusses the TEDS APIs in detail. Note: There is no requirement to use the TEDS API and developers can elect to use the ANSI SQL interface to the database.

3.4.3.4 Operating System File System Services

The operating system file system is used by the RDBMS to implement the relational database.

PRELIMINARY

3.5

Performance Tradeoff Analysis

Two database architectures using relational databases were prototyped to examine the trade-off between functionality and performance. Naval METOC databases have been implemented using the binary large object (BLOB) data type as the primary way to store METOC information. This approach optimizes the speed of information storage and retrieval, at the expense of reduced functionality. An alternative method would be to store each attribute of a record explicitly in tables. This would make better use of the database's functions, but it would also slow down the storage and retrieval processes. A main task of this study is to determine if there is a satisfactory architectural compromise between the two techniques.

For the prototype comparison, data structures and tables were constructed using a BATHY observation report (WMO Manual on Codes Form FM-63). Subroutines providing identical interfaces to the database were implemented for each design.

The first type of architecture contains a two-table format that provides a column for each field of the report. One table holds station, sea-surface, and basic report data; the other table contains water depth and temperature readings at a variable number of levels below the surface. These tables are joined, or linked, by the WMO region/sub-region column index, a unique integer used to associate a given row of report data in the first table with its corresponding “levels” data in the second. The second design incorporates all information in one table with columns for latitude, longitude, and time, and a BLOB combining all of the BATHY data in a byte format.

Several tests were devised to gauge the performance of the two prototypes. These trials executed the following typical database operations:

- Adding data
- Getting a summary (catalog) of available records
- Retrieving records based on latitude, longitude, and time selection criteria
- Retrieving data meeting certain observable conditions found in a record
- Updating records.

PRELIMINARY

The tests were run under multiple environments on a nominally loaded system, under HP-UX 10.10 using Informix RDBMS Version 7.2. The three basic scenarios involved accessing a database on a local hard disk, via a local-area network (LAN), or via a wide-area network (WAN). Before each of these tests, the database server was re-initialized and the database dropped and re-created.

As part of the tests, two methods of making disk space available were also compared. One was raw disk space, in which no file system is associated with a disk partition. Thus, the UNIX file-management subsystem is bypassed. This was contrasted with the cooked file method, which uses the UNIX file-management system. Data is placed into the UNIX kernel's buffer before being sent to the appropriate memory location. Experiments were also conducted on two raw-environment variations using a redundant array of inexpensive disks (RAID) -- a hardware mirroring method -- and a non-RAID disk containing a database accessed through a LAN. The raw technique, according to Informix documentation, is said to provide "orders of magnitude" better performance because it transfers data directly from disk to the application using the computer's direct memory access hardware.

The same set of data was used to test each design, with locations distributed regularly between 32 degrees north and south latitude and 64 degrees east and west longitude. Initially, ten records were inserted into empty tables. Next, queries were performed to retrieve a catalog of, and details of, these records. The same tests were then applied to sets of 100, 1,000, and 10,000 records. For the initial architecture comparison over a 10,000-record set, additional queries were made over areas ranging from 1 by 1 to 60 by 60 degrees. For the environment tests, retrievals and updates over multiple records were simulated by performing these actions repeatedly on a single record (1, 10, 100, and 1,000 times).

The results for adding records show that under both architectures, the time required increases linearly with the number of records being inserted. Filling the single-table (BLOB) tables is faster by a factor of 2 to 3 than filling the tables in the two-table (columnar) architecture (0.03 - 45 seconds for the BLOB table, 0.1 to 100 seconds for the columnar tables). This is to be expected because of the

PRELIMINARY

extra overhead incurred through use of another table, and because the data must be broken out before being stored in the columnar table.

Catalog retrieval times were similar for both architectures, ranging from 0.01 to 2.6 seconds. This is also expected because the query criteria (a time range and an area defined by latitude and longitude limits) can be tested against data for which columns are explicitly provided (and in one table) for both designs.

Time differences were especially noticeable when retrieving detailed records. For simple retrievals, which obtain all records in a specified area, the BLOB format is much faster than the fully columnar design (0.02 - 4.7 seconds for the BLOB format, 0.06 - 57 seconds for the columnar design). This was particularly evident when retrieving a large number of records. The query criterion (location) for this type of retrieval is similar to that used for compiling a catalog, for which the times were nearly equal. Therefore, the difference in times is considered to result from the fact that, in the BLOB architecture, the record can be retrieved by consulting only a few columns in a single table, while in the columnar architecture it is necessary to consult many columns in two tables.

When conditional retrievals based on examination of values contained within the observation data were performed, however, the columnar architecture proved to be faster than the BLOB architecture by about a factor of 3 (0.1 - 1.5 seconds for the columnar architecture, 0.1 - 4.5 seconds for the BLOB architecture). SQL methods can be applied directly to the data in the two-table design in order to build a report, while each BLOB must be opened using an API before criteria can be tested and records meeting the conditions found. The major drawback of using this method is that the RDBMS functionality has been bypassed, and equivalent proprietary functionality would have to be developed.

A comparison between the raw and cooked environments showed that under both architectures and for all database operations, the raw method was only slightly faster for databases up to 1000 records in size. Neither adjusting configuration-file parameters nor dropping indices changed this outcome. Consultations with Informix's technical-support staff revealed that only database sizes on the order of 1 GB (approximately 7.5 million records in this case) would produce the expected

PRELIMINARY

significant improvement. However, the results using a 10,000-record (~ 1 MB) database seem to contradict this expectation; retrieval and update times using the raw technique actually *grow* to double and triple the cooked times.

Tests using a non-RAID disk yielded times that, in general, were 5-10% greater than those found with a RAID disk and about the same as the cooked case's results. The LAN test followed a pattern similar to that of the non-RAID test, with the times increasing slightly as operations were performed on larger databases and a higher number of records, requiring more time to transfer data across the network. The WAN test, connecting computers in Monterey and Newport, showed, though varying network loads at different times of the day probably affected the exact times.

An interesting result from the environment tests that did not use a network was that the two-table design was as fast as or slightly faster than over the BLOB method in retrieving from and updating the 10,000-record database, indicating that the environments were optimized to allow the two-table format to work more efficiently for large databases.

The results on the following pages compare the performance of the two designs for typical database operations (add, catalog, detail, retrieve, and update) and under various environments. The catalog and detail actions apply to all records (i.e., 10, 100, 1,000, 10,000) added.

The following table is a comparison of the elapsed wall time (in seconds) taken to add records to an empty database.

Add:	Count	Blob	Table
	10	0.081937	0.098608
	100	0.339790	0.931072
	1000	3.540332	12.310156
	10000	41.327833	99.191061

PRELIMINARY

This table compares the time taken to build a catalog of the contents of the database for all records that were added (above) and, in the 10,000 record case, those that are in a search rectangle of size (degree x degree).

Catalog:	Count/Size	Blob	Table
	10	0.018172	0.017762
	100	0.040350	0.038876
	1000	0.270731	0.274549
	10000	2.680395	2.614344
	18 / 1x1	1.157718	1.089371
	242 / 5x5	1.214617	1.135658
	882 / 10x10	1.326052	1.233727
	2792 / 20x20	1.628811	1.556338
	6928 / 40x40	2.287387	2.171683
	9488 / 60x60	2.649785	2.540826

This table compares the time taken to build a list of bathy reports from contents of the database for all records that were added and, in the 10,000 record case, those that are in a search rectangle.

Details:	Count/Size	Blob	Table
	10	0.017492	0.067814
	100	0.055588	0.549665
	1000	0.447961	5.512118
	10000	4.564749	54.399264
	18 / 1x1	1.173762	1.175740
	242 / 5x5	1.261422	2.419819
	882 / 10x10	1.505349	5.787298
	2792 / 20x20	2.163797	16.006008
	6928 / 40x40	3.521866	38.268367
	9488 / 60x60	4.292004	52.412481

PRELIMINARY

This table lists times from a cooked environment test:

Records	BLOB			2-Table		
	Add	Catalog	Detail	Add	Catalog	Detail
10	0.08	0.02	0.02	0.14	0.02	0.09
100	0.49	0.04	0.09	1.39	0.04	0.72
1000	5.82	0.28	0.73	15.97	0.27	7.20
10000	73.80	2.67	2.67	153.78	2.57	73.30

10 Records:

Iterations	Retrieve		Update	
1	0.01	0.01	0.01	0.02
10	0.05	0.06	0.13	0.15
100	0.46	0.54	1.28	1.51
1000	4.66	5.60	12.68	15.46

100 Records:

Iterations	Retrieve		Update	
1	0.01	0.01	0.02	0.02
10	0.07	0.09	0.14	0.18
100	0.61	0.80	1.39	1.72
1000	5.98	8.06	13.98	17.38

1000 Records:

Iterations	Retrieve		Update	
1	0.03	0.04	0.03	0.05
10	0.20	0.36	0.26	0.42
100	2.00	3.52	2.64	4.19
1000	19.52	34.62	25.34	44.27

PRELIMINARY

10000 Records:

Iterations	Retrieve	Update	Retrieve	Update
1	0.29	0.44	0.28	0.05
10	2.55	3.99	2.38	3.69
100	25.05	39.51	23.59	36.47
1000	251.87	396.10	240.36	362.56

This table shows times using a raw, non-RAID disk:

Records	Blob			2-Table		
	Add	Catalog	Detail	Add	Catalog	Detail
10	0.08	0.02	0.02	0.15	0.02	0.09
100	0.55	0.04	0.08	1.51	0.04	0.71
1000	5.66	0.30	0.78	15.55	0.26	7.03
10000	56.91	2.92	9.18	151.57	2.86	71.68

10 Records:

Iterations	Retrieve	Update	Retrieve	Update
1	0.01	0.01	0.01	0.02
10	0.05	0.06	0.13	0.15
100	0.48	0.55	1.27	1.51
1000	4.64	5.51	12.55	15.10

100 Records:

Iterations	Retrieve	Update	Retrieve	Update
1	0.01	0.01	0.02	0.02
10	0.06	0.08	0.14	0.17
100	0.60	0.80	1.37	1.74
1000	6.03	7.99	13.78	17.41

PRELIMINARY

1000 Records:

Iterations	Retrieve	Update	Retrieve	Update
1	0.04	0.04	0.05	0.05
10	0.20	0.35	0.26	0.42
100	1.90	3.38	2.51	4.12
1000	18.94	33.84	24.90	42.04

10000 Records:

Iterations	Retrieve	Update	Retrieve	Update
1	1.23	1.22	0.96	0.81
10	5.60	6.47	6.46	7.04
100	54.12	64.80	63.10	99.53
1000	769.34	825.91	791.97	873.36

This table details the times found from a raw, RAID disk test:

Records	Blob			2-Table		
	Add	Catalog	Detail	Add	Catalog	Detail
10	0.07	0.02	0.02	0.17	0.02	0.07
100	0.46	0.04	0.06	1.22	0.04	0.59
1000	5.59	0.36	0.61	13.91	0.34	5.79
10000	71.65	4.38	7.01	147.30	4.10	61.70

10 Records:

Iterations	Retrieve	Update	Retrieve	Update
1	0.01	0.01	0.01	0.01
10	0.04	0.05	0.11	0.13
100	0.36	0.46	1.05	1.24
1000	3.54	4.56	10.55	12.21

PRELIMINARY

100 Records:

Iterations	Retrieve	Update	Retrieve	Update
1	0.01	0.01	0.01	0.02
10	0.05	0.07	0.12	0.15
100	0.49	0.72	1.17	1.49
1000	4.92	7.24	11.60	14.89

1000 Records:

Iterations	Retrieve	Update	Retrieve	Update
1	0.04	0.04	0.03	0.04
10	0.19	0.35	0.24	0.39
100	1.84	3.37	2.30	3.87
1000	18.35	33.58	22.87	38.61

10000 Records:

Iterations	Retrieve	Update	Retrieve	Update
1	0.93	0.93	0.93	0.93
10	8.21	8.99	7.87	8.60
100	82.53	91.67	76.01	86.82
1000	812.57	918.09	777.53	865.34

This table shows times from a raw, LAN test:

Records	Blob			2-Table		
	Add	Catalog	Detail	Add	Catalog	Detail
10	0.06	0.02	0.02	0.17	0.02	0.09
100	0.52	0.10	0.12	1.54	0.04	0.73
1000	5.69	0.44	1.08	16.19	0.29	7.37
10000	74.10	5.23	17.42	177.26	2.79	81.96

PRELIMINARY

10 Records:

Iterations	Retrieve	Update	Retrieve	Update
1	0.01	0.01	0.01	0.02
10	0.06	0.06	0.14	0.16
100	0.56	0.60	1.36	1.55
1000	5.65	6.13	13.47	15.58

100 Records:

Iterations	Retrieve	Update	Retrieve	Update
1	0.01	0.01	0.02	0.02
10	0.06	0.08	0.15	0.18
100	0.63	0.80	1.43	1.77
1000	6.29	8.03	14.36	17.82

1000 Records:

Iterations	Retrieve	Update	Retrieve	Update
1	0.03	0.04	0.04	0.05
10	0.19	0.34	0.26	0.41
100	1.92	3.34	2.53	4.12
1000	19.00	33.59	25.42	44.34

10000 Records:

Iterations	Retrieve	Update	Retrieve	Update
1	0.84	0.92	0.75	0.80
10	8.95	8.28	6.89	15.29
100	87.31	85.25	59.46	72.15
1000	718.31	780.37	839.36	1064.68

In summary, the single-table architecture consolidates the observation data in a single column, while the two-table format stores each data field explicitly. The BLOB design limits the variety of possible queries but provides faster retrieval for

PRELIMINARY

queries based only on location and time. More complex queries that involve examination of the observation data place this method at a disadvantage. The two-table technique is slower in inserting records and in retrieving records based only on location and time, but provides more flexibility in queries and is faster in retrieving data based on complex queries. Though Informix manuals state that cooked files degrade performance, and Informix's technical-support staff said that the raw method would produce significantly better results, especially with large databases, tests in a variety of environments showed little time advantage from using the raw technique instead of the cooked approach. Further study of the raw method, with databases of around 1 GB, seems to be warranted.

There are several ways to take advantage of the utility that a database would provide in storing grid data, though all would involve more overhead than is now used by storing the data exclusively as blobs. One method would be to define a table for a given level, having columns for commonly referenced parameters.

Tests have been run to determine the feasibility of using such a table to hold grid data instead of keeping it in the BLOB format. The tests were based on a table that has seven columns: location (latitude/longitude values), dew point, height, air temperature, north-south and east-west components of wind velocity, and origin (valid time and forecast period) of the gridded data.

Two techniques were tried to ingest grids into this table. It was assumed that grids would be received from a decoder with an array of values in a known order of geographic locations. In the first test, when a new grid was received, the API selected all records in the table for that grid's origin. From those records, an array of values that contained the queried fields was built. These records were then deleted from the database, the new grid was stored in the RAM-resident array, and this data was written back to the database. This selection, deletion, and insertion routine was repeated, updating one column at a time, until each of the columns (dew point, height, temperature, and wind velocity components) was filled. The second method performed the same process as above except that the query deletion and reinsertion steps were replaced by an update cursor.

PRELIMINARY

One set of results was produced using a table with indices turned off until after a complete grid was retrieved; the second set included indices throughout the ingest process. Both sets started with an empty table, to which 10,512 records were added and times found. The record count was incremented repeatedly by 10,512, and times measured, until each table held 105,120 records (from a 73 x 144 grid).

For the non-indexed test, the total update time for each column consistently fell in the 8 to 13 second range for all record counts, with about 0.6 seconds devoted to populating the arrays, 2 to 5 seconds to deleting records, and 4 to 6 seconds to reinserting records. Using the update cursor method, the time increased to between 23 and 28 seconds over all record counts. After the table was filled with 105,120 records and indices were added, the update time rose to between 50 and 60 seconds without a cursor, but decreased to 21 to 24 seconds with a cursor.

With indices turned on before filling the tables, insertion and deletion times increased approximately linearly with the number of records, from approximately 7 to 37 seconds, and the update time rose from about 16 to 75 seconds. When updating was done with a cursor, however, the time stayed between 19 and 23 seconds per column for all record counts.

It appears from the test results that turning indices off until grid data retrieval is completed provides the more efficient method of updating the table. Keeping the indices on may have a slight advantage when a cursor is used, but in that case, the order in which data is updated cannot be determined in advance.

Assuming that each table would contain forecasts out to 120 hours in 12-hour increments, it could be anticipated that there would be eleven tables each having 400 sets of records, 20 of which would be replaced every 12 hours. In such a case, for instance, over 3 hours of wall time would be required to ingest 1100 NOGAPS grids, versus about 15 minutes for storing them as blobs. In addition, there would be a 40% increase in space requirements. However, the time period could easily be cut by 80% by using a preprocessor which would collect the record groups before the API was called to ingest them, so that a single insert instead of five updates could be used. Other management techniques or alternative schema could provide additional time and space savings.

PRELIMINARY

Based on the test results, a hybrid schema, which would keep the heavily used data in a tabular form and the less frequently used data as BLOB records, could be a feasible way of storing grid data. Among the possible advantages would be rapid retrieval of only the specific data necessary for a process or product and the ability to get gridded data without necessarily employing an API. Other considerations would include whether or not to write additional code or how to construct tables to reduce space and time overhead, and how to structure tables that would be practical for an end user.

The following tables show results when using a table (without indices) to store grid data. Each insert and update involves 10,512 records. Before the first test, the table is empty. After the table has been filled with 105,120 records, a test performing operations on 10,512 of those records is done. The last test shows the times for a 105,120-record database after indices are added to the table. (All times in seconds)

(Before insert: 0 records

After insert: 10,512 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.01	0.01	3.09	3.13	25.08	Dew Point
0.65	1.48	5.13	9.41	27.99	Height
0.63	1.46	4.09	7.54	25.74	Temperature
0.63	1.49	4.19	7.56	21.80	Wind U
0.56	1.43	4.73	8.31	24.24	Wind V

(Before insert: 10,512 recordsAfter insert: 21,024 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.01	0.31	7.75	8.41	23.27	Dew Point
0.61	4.41	5.14	11.60	22.84	Height
0.63	3.32	3.98	9.29	25.98	Temperature
0.64	1.93	5.01	8.71	26.14	Wind U
0.61	3.07	8.30	13.19	26.10	Wind V

PRELIMINARY

(Before insert: 21,024 records After insert: 31,536 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.01	0.55	5.57	6.56	26.14	Dew Point
0.57	4.57	5.15	11.61	27.66	Height
0.64	2.25	4.98	9.22	27.95	Temperature
0.64	3.10	6.68	11.66	27.72	Wind U
0.65	2.96	5.74	10.63	26.61	Wind V

(Before insert: 31,536 records After insert: 42,048 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.01	0.84	5.46	7.06	23.01	Dew Point
0.63	2.47	5.40	10.07	26.59	Height
0.64	3.05	8.42	13.53	28.14	Temperature
0.64	5.09	6.11	13.38	23.21	Wind U
0.63	3.31	4.55	10.08	26.22	Wind V

(Before insert: 42,048 records After insert: 52,560 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.01	1.18	5.38	7.45	23.68	Dew Point
0.64	3.55	4.87	11.09	27.07	Height
0.60	3.26	5.90	12.22	23.81	Temperature
0.64	4.33	5.21	11.92	27.13	Wind U
0.58	3.99	4.72	10.87	25.25	Wind V

(Before insert: 52,560 records After insert: 63,072 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.01	1.43	6.49	9.06	25.17	Dew Point
0.64	4.52	6.66	13.76	26.21	Height

PRELIMINARY

0.65	3.78	4.84	11.11	26.90	Temperature
0.65	5.50	6.02	13.92	26.96	Wind U
0.65	3.12	6.35	11.93	29.06	Wind V

(Before insert: 63,072 records After insert: 73,584 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.01	1.74	5.50	8.62	25.71	Dew Point
0.63	5.91	5.07	13.72	27.07	Height
0.64	3.94	4.36	11.13	28.66	Temperature
0.63	3.72	4.57	10.89	29.48	Wind U
0.64	3.91	6.18	13.08	28.56	Wind V

(Before insert: 73,584 records After insert: 84,096 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.01	2.01	6.72	10.22	28.16	Dew Point
0.63	6.61	4.27	13.89	30.85	Height
0.64	3.92	6.41	13.34	28.53	Temperature
0.64	5.52	7.14	15.59	26.51	Wind U
0.64	4.61	4.62	12.23	27.24	Wind V

(Before insert: 84,096 records After insert: 94,608 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.01	2.16	5.93	9.87	25.66	Dew Point
0.64	3.80	4.33	11.53	26.09	Height
0.56	4.24	7.33	14.97	28.65	Temperature
0.64	4.84	4.47	12.49	27.84	Wind U
0.63	4.04	4.12	11.30	30.18	Wind V

PRELIMINARY

(Before insert: 94,608 records After insert: 105,120 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.01	2.42	4.91	9.22	26.51	Dew Point
0.58	5.15	4.75	13.33	29.12	Height
0.64	4.29	4.81	12.55	28.19	Temperature
0.64	5.77	4.19	13.29	30.20	Wind U
0.65	5.99	5.08	14.44	26.89	Wind V

(Repeat first run, this time with 105,120 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.66	4.27	4.71	12.39	23.72	Dew Point
0.63	4.86	5.77	14.07	27.32	Height
0.64	6.61	5.70	15.49	27.31	Temperature
0.64	3.93	5.27	12.58	29.55	Wind U
0.58	4.54	5.62	13.77	31.19	Wind V

(Repeat the above run after adding indices)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.63	25.60	29.68	57.94	20.93	Dew Point
0.65	23.18	23.76	50.00	23.17	Height
0.56	26.54	32.87	62.21	23.73	Temperature
0.64	21.10	28.67	52.40	24.05	Wind U
0.66	23.51	36.27	62.59	22.88	Wind V

The following tables show results when using a table (with indices) to store grid data. Each insert and update involves 10,512 records. Before the first test, the table is empty. After the table has been filled with 105,210 records, a test performing operations on 10,512 of those records is done. (All times in seconds)

PRELIMINARY

(Before insert: 0 records)

After insert: 10,512 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.01	0.01	7.10	7.13	23.00	Dew Point
0.66	6.89	7.08	16.65	20.05	Height
0.61	7.55	7.55	17.91	21.09	Temperature
0.66	7.77	6.30	16.90	20.10	Wind U
0.64	7.35	6.01	16.07	20.90	Wind V

(Before insert: 10,512 recordsAfter insert: 21,024 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.01	0.01	9.32	9.33	20.66	Dew Point
0.64	11.62	12.65	26.90	24.50	Height
0.66	11.01	10.70	24.48	21.97	Temperature
0.65	11.04	11.39	25.53	25.56	Wind U
0.64	13.38	12.00	28.11	22.48	Wind V

(Before insert: 21,024 recordsAfter insert: 31,536 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.01	0.01	21.08	21.10	20.46	Dew Point
0.63	14.16	13.95	30.63	22.27	Height
0.65	12.00	16.65	31.48	21.41	Temperature
0.66	14.74	11.80	29.88	23.43	Wind U
0.65	13.10	13.21	29.06	24.97	Wind V

(Before insert: 31,536 recordsAfter insert: 42,048 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.01	0.01	21.34	21.36	24.52	Dew Point
0.64	18.02	20.13	40.75	21.39	Height

PRELIMINARY

0.64	19.84	19.35	42.17	19.83	Temperature
0.64	23.49	27.81	54.23	19.84	Wind U
0.61	23.10	17.87	43.80	25.43	Wind V

(Before insert: 42,048 records After insert: 52,560 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.01	0.01	25.16	25.18	24.24	Dew Point
0.64	15.28	23.21	41.20	22.39	Height
0.64	21.64	22.01	46.47	20.65	Temperature
0.64	21.14	25.65	49.96	24.77	Wind U
0.66	21.03	18.39	42.50	25.37	Wind V

(Before insert: 52,560 records After insert: 63,072 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.01	0.01	22.46	22.48	23.65	Dew Point
0.64	17.59	18.74	38.93	22.63	Height
0.65	19.60	19.64	42.08	21.37	Temperature
0.63	14.16	21.72	38.67	23.07	Wind U
0.64	22.59	16.90	42.24	21.09	Wind V

Before insert: 63,072 records After insert: 73,584 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.01	0.01	22.46	22.47	27.55	Dew Point
0.65	15.08	19.75	37.48	25.66	Height
0.63	20.06	20.54	43.31	25.35	Temperature
0.64	18.82	18.62	40.03	23.82	Wind U
0.64	16.56	23.67	42.93	25.96	Wind V

PRELIMINARY

Before insert: 73,584 records After insert: 84,096 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.01	0.01	24.75	24.78	22.08	Dew Point
0.64	15.20	19.15	37.05	22.71	Height
0.65	19.54	19.14	41.66	21.71	Temperature
0.64	17.26	21.92	42.13	23.00	Wind U
0.65	19.63	17.20	39.93	23.92	Wind V

(Before insert: 84,096 recordsAfter insert: 94,608 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.01	0.01	29.58	29.59	19.43	Dew Point
0.65	29.32	36.49	69.12	22.01	Height
0.62	33.06	32.80	68.93	22.79	Temperature
0.64	29.77	27.33	60.02	21.23	Wind U
0.64	26.30	23.88	53.10	23.53	Wind V

(Before insert: 94,608 recordsAfter insert: 105,120 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.01	0.01	35.52	35.60	21.01	Dew Point
0.64	32.30	37.95	73.03	22.95	Height
0.59	37.34	37.52	77.88	21.62	Temperature
0.60	41.52	39.53	84.11	25.20	Wind U
0.64	36.28	35.14	74.42	24.11	Wind V

(Repeat first run, this time with 105,120 records)

Select	Delete	Insert	Regular Update	Cursor Update	Column
0.66	33.28	34.28	70.67	25.63	Dew Point
0.63	38.37	35.03	76.48	23.29	Height

PRELIMINARY

0.63	31.35	29.60	63.82	27.54	Temperature
0.64	34.56	34.41	71.81	26.00	Wind U
0.63	31.42	29.53	63.62	22.89	Wind V

The TEDS API returns multiple records as a linked list. There has been some concern that the overhead involved in creating these lists could be impeding performance.

To evaluate this concern, two types of retrievals were performed. The first obtained a catalog of records and the second, a set of complete BATHY records found within areas of varying size; groups of up to 10,000 records were retrieved. Each of the full records contained an array of levels. The sets were built two ways, using a linked list or an array.

Four comparisons were run between the array and linked-list approaches. The first of these determined the finished size of the arrays by first querying the database for a count of the records before doing the actual retrievals. Two other tests started by allocating a fixed block of memory and adding successive increments as necessary. A final reallocation was then performed to free any excess memory. The fourth test used a large (2 MB) static memory block set aside to provide more than enough room for the expected maximum number of records.

Test results indicated that the API is not being adversely affected by the use of linked lists. For the first test, the array technique took nearly double the time to get a catalog of records, and at least twice as long to retrieve details of records, indicating that it is probably as much work for the database to count the number of qualifying records as it is to return them. For the other three tests, the array approach yielded times slightly (less than 1%) better than linked lists.

Converting the API to provide arrays instead of linked lists is therefore not recommended, not only because there does not appear to be any appreciable gain in efficiency, but also because such a conversion would not be upwardly compatible with any existing linked-list processes. There would also be potential

PRELIMINARY

portability problems requiring a more complex interface to insure platform independence.

The table below compares the linked-list (LL) and array approaches to returning multiple records from a 10,000-record database, using a RAID disk and a cooked file. Catalog and detail retrieval times (in seconds) are shown for various location box sizes (degree x degree) and the corresponding number of observations found in each box. The “Database count” column shows the results from doing a count of records before doing retrievals. The “8 record” and “256 record” columns indicate tests done with initial memory allocations equivalent to the size of 8 and 256 records, respectively. The “2,048 bytes” and “32,768 bytes” columns illustrate tests with allocations of these sizes. The “Static array” column indicates the results of tests done using a unchanging 2 MB memory block.

PRELIMINARY

RAID, Cooked	Database count	8 records	256 records	2,048 bytes	32,768 bytes	Static array	Method
1X1 (18 records)	1.07	1.08	1.07	1.07	1.07	1.06	LL catalog
	2.13	1.09	1.08	1.08	1.08	1.06	Array catalog
	1.17	1.18	1.17	1.17	1.18	1.16	LL detail
	2.28	1.17	1.17	1.17	1.17	1.16	Array detail
2X2 (50 records)	1.07	1.07	1.07	1.07	1.07	1.07	LL catalog
	2.12	1.07	1.07	1.07	1.07	1.06	Array catalog
	1.35	1.35	1.36	1.36	1.36	1.34	LL detail
	2.57	1.36	1.35	1.36	1.36	1.35	Array detail
10X10 (882 records)	1.21	1.20	1.20	1.20	1.20	1.20	LL catalog
	2.29	1.19	1.20	1.20	1.20	1.19	Array catalog
	6.04	6.07	6.02	6.11	6.07	6.03	LL detail
	10.25	6.00	5.97	5.99	5.98	6.01	Array detail
30X30 (5,382 records)	1.85	1.86	1.86	1.86	1.86	1.85	LL catalog
	3.03	1.84	1.82	1.83	1.82	1.84	Array catalog
	31.52	31.37	31.18	31.12	31.13	31.24	LL detail
	51.40	32.14	31.40	31.05	30.99	30.73	Array detail
65X65 (10,000 records)	2.57	2.49	2.54	2.50	2.52	2.53	LL catalog
	3.77	2.49	2.46	2.47	2.42	2.45	Array catalog
	56.57	56.81	56.75	56.41	56.50	59.16	LL detail
	93.20	56.77	56.45	56.94	56.73	56.50	Array detail

PRELIMINARY

This table compares the linked-list (LL) and array approaches to returning multiple records from a 10,000-record database, using a RAID, raw disk environment. Catalog and detail retrieval times (in seconds) are shown for various location box sizes (degree x degree) and the corresponding number of observations found in each box. The “Database count” column shows the results from doing a count of records before doing retrievals. The “8 record” and “256 record” columns indicate tests done with initial memory allocations equivalent to the size of 8 and 256 records, respectively. The “2,048 bytes” and “32,768 bytes” columns illustrate tests with allocations of these sizes. The “Static array” column indicates the results of tests done using a unchanging 2 MB memory block.

RAID, Raw	Database count	8 records	256 records	2,048 bytes	32,768 bytes	Static array	Method
1X1 (18 records)	2.39	2.35	2.37	2.32	2.34	2.35	LL catalog
	3.80	2.28	2.33	1.51	2.29	2.29	Array catalog
	1.69	1.67	1.64	1.62	1.68	1.68	LL detail
	3.48	1.61	1.61	1.60	1.60	1.60	Array detail
2X2 (50 records)	1.96	1.90	1.48	1.91	1.47	1.93	LL catalog
	3.10	1.77	1.47	1.59	1.47	1.77	Array catalog
	1.95	1.82	2.26	2.03	2.24	1.85	LL detail
	4.28	1.80	1.95	1.84	2.14	1.80	Array detail
10X10 (882 records)	1.64	2.44	1.80	1.61	2.17	1.60	LL catalog
	3.09	1.94	1.64	2.12	1.92	2.51	Array catalog
	6.92	6.75	6.72	7.14	7.40	6.74	LL detail
	12.23	6.64	7.15	7.68	6.87	6.72	Array detail
30X30 (5,382 records)	3.03	2.75	2.42	3.12	2.26	3.01	LL catalog
	3.99	2.43	2.93	2.22	2.80	2.56	Array catalog

PRELIMINARY

RAID, Raw	Database count	8 records	256 records	2,048 bytes	32,768 bytes	Static array	Method
	32.58	32.96	32.67	32.93	32.56	33.02	LL detail
	53.22	32.48	31.98	32.10	32.38	32.38	Array detail
65X65 (10,000 records)	3.59	3.60	3.33	3.61	3.71	3.50	LL catalog
	4.62	2.98	3.23	2.96	2.83	3.06	Array catalog
	60.08	60.25	59.38	59.51	60.72	59.68	LL detail
	94.73	59.11	59.04	58.87	59.78	59.16	Array detail

PRELIMINARY

4

DETAILED DESIGN OF THE DATABASE

The following paragraphs describe the METOC Database in terms of its component database segments. As described in Section 3, the METOC Database has been broken out into 6 shared database segments. Because the segments are intended to be functionally independent shared database segments, conceptual level, logical level, and physical level designs are presented for each of the segments.

The figures in this section show the symbology used in the entity-relationship diagrams in the subsections that follow. Figure 4-1 shows the symbols used to describe relationships. Figure 4-2 shows the parent and child entity definitions in the physical level models. Figure 4-3 shows the relationships used in the logical level models.

PRELIMINARY

This figure depicts the symbols used in the physical data and logical data models for Model Data, Observation Data, Remotely Sensed Data, Text Observations, and Images.

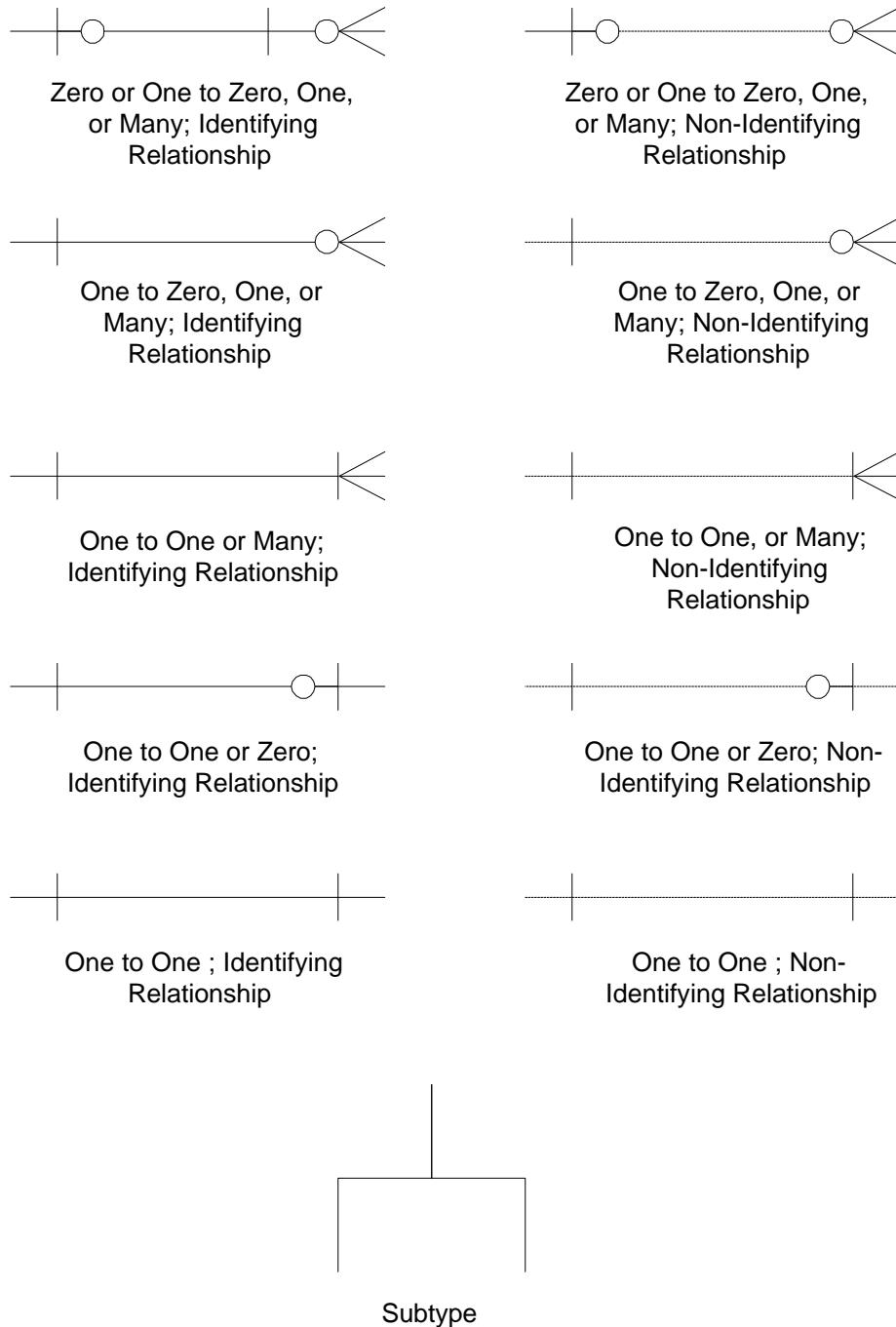


Figure 4-1. Symbology Used in Entity-Relationship Diagrams

PRELIMINARY

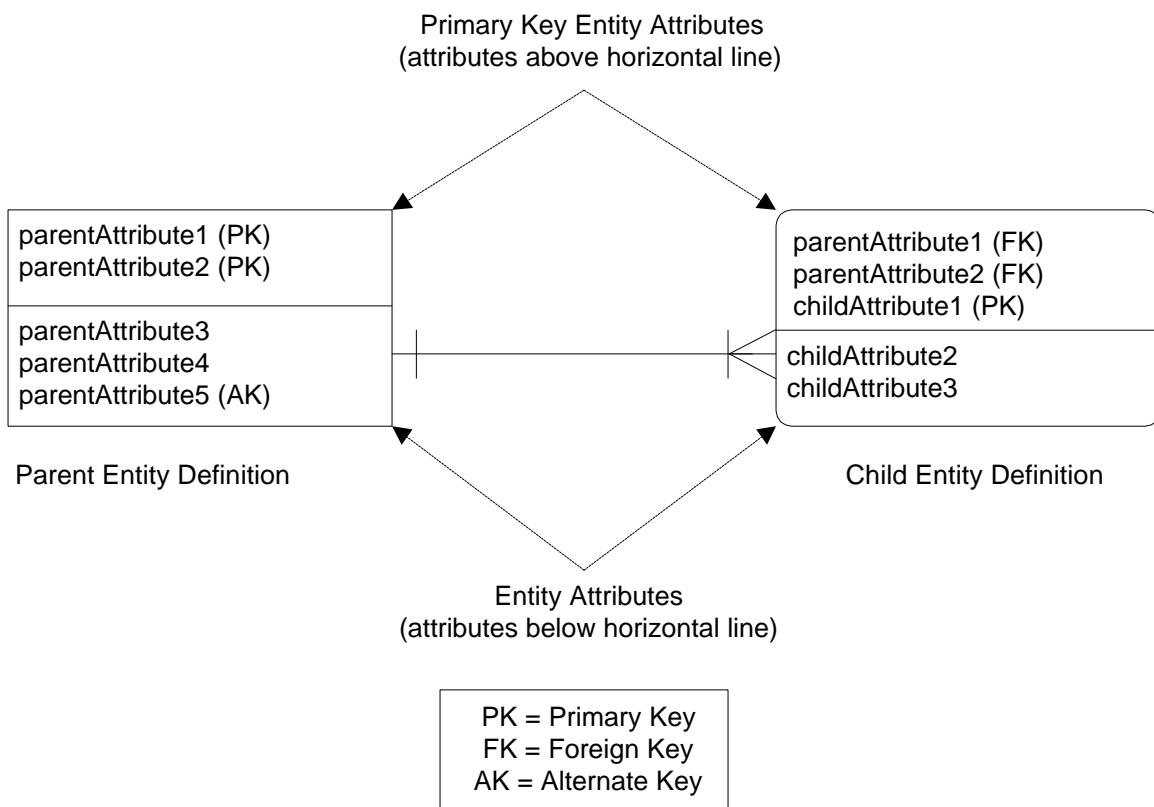


Figure 4-2. Entity Conventions for Physical Models

PRELIMINARY

Attributes inherited from a parent and public or
protected attributes defined in sub-class
(attributes above horizontal line)

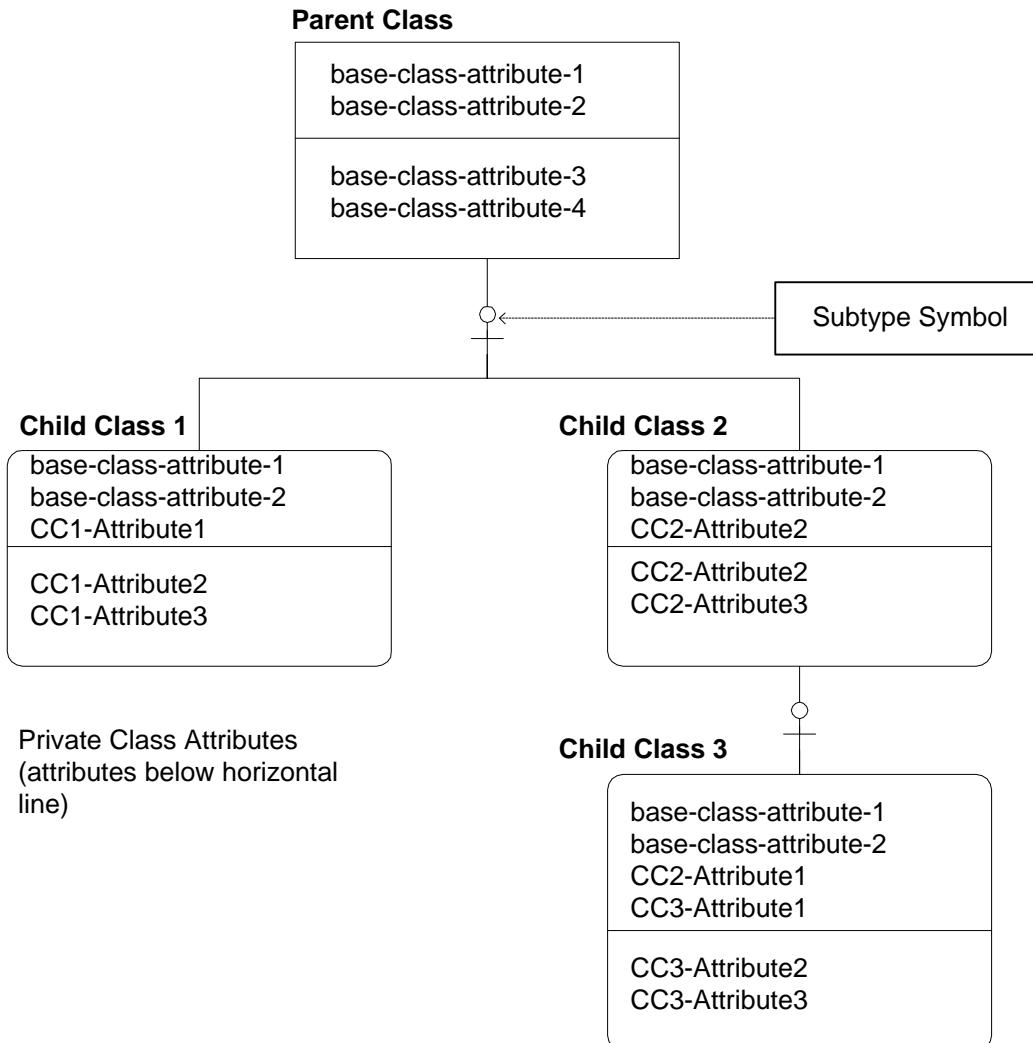


Figure 4-3. Logical Model E-R Diagram Conventions

PRELIMINARY

4.1 Grid Field Data Segment (MDGFD) Design

4.1.1 MDGFD Conceptual Level Design

Grid field data is received from the FLTALT CSCI and stored in MDGFD as datasets organized by data type, time, and geographic area. A dataset is a logical collection of grid field records where each record represents a set of homogeneous grid element values (e.g., temperature), at a particular level (e.g., 500 MB), and for a specific forecast period (e.g., 24 hours). Conceptually, there are two tables used to organize the datasets: the *grid dataset directory* table and the *grid dataset detail* table.

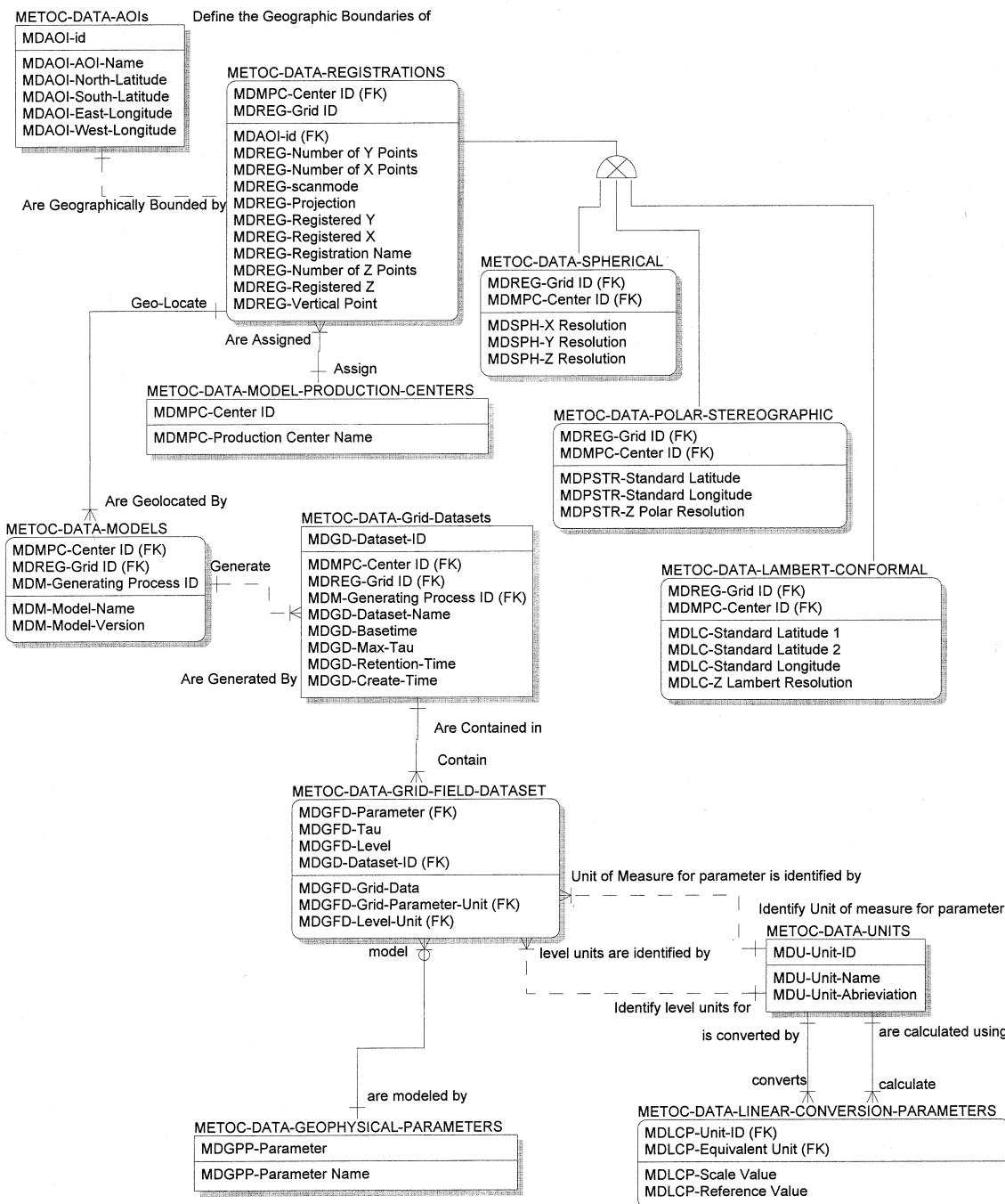
The grid dataset directory table maintains descriptive information about the datasets and supports catalog type queries. Each entry (or row) in the directory table provides descriptive information for a single dataset detail table. Descriptive information includes grid model name, basetime associated with the grid dataset, and geographical bounds of the grid dataset.

The grid dataset detail table represents a logical collection of discrete grid field data records and stores additional descriptive information about the individual geophysical data records stored in the dataset. These data records are logically associated with each other by grid model type and base time. A detail table row, for example, would store the descriptive information about a specific grid field parameter (e.g., Temperature), at a specific level (e.g., 500 MB), for a specific forecast period (e.g., 24z). The detail table row also contains the actual geophysical grid values.

To facilitate access both on ingest and retrieval, the geophysical grid values are stored as Binary Large Objects (BLOB).

PRELIMINARY**4.1.2 MDGFD Logical Level Design**

The entity-relationship diagram in Figure 4.1-1 on the next page depicts the logical model of the Grid Field Data Segment.

**Figure 4.1-1. Logical Level Design of the MDGFD**

PRELIMINARY

4.1.3 MDGFD Physical Level Design

Figure 4.1-2 presents the physical level design of the MDGFD in entity-relationship diagrams.

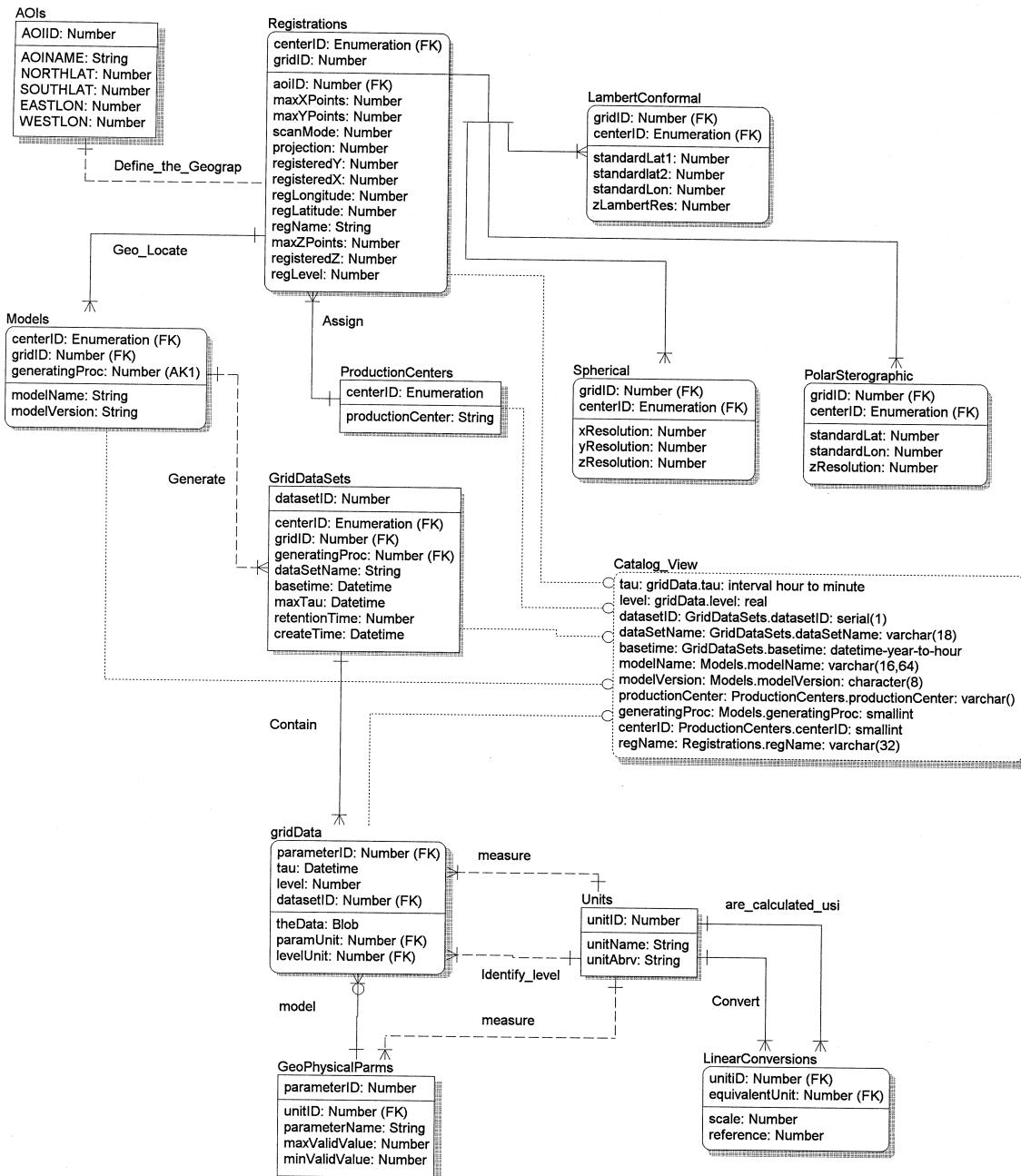


Figure 4.1-2. Physical Level Design of the MDGFD

PRELIMINARY

The remainder of this section presents the designs of the individual tables that make up the MDGFD.

4.1.3.1 *Area of Interest Table*

Table Name: AOIs

Description: Names of areas of interest mapped to a unique ID and type.

Primary Key: AOIID

Table 4.1-1. AOIs Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
AOIID	serial(1)	NOT NULL	1	2^{32}	Unique numeric identifier for the AOI.
AOINAME	varchar(64)	NOT NULL	N/A	N/A	Alpha/Numeric name of AOI.
EASTLON	real	NOT NULL	-180.0	180.0	Easternmost bounding vector of AOI.
NORTHLAT	real	NOT NULL	-90.0	90.0	Northernmost bounding vector of AOI.
SOUTHLAT	real	NOT NULL	-90.0	90.0	Southernmost bounding vector of AOI.
WESTLON	real	NOT NULL	-180.0	180.0	Westernmost bounding vector of AOI.

PRELIMINARY

4.1.3.2 *Geophysical Parameters Table*

Table Name: GeoPhysicalParms

Description: Normalized table storing information about environmental parameters storable in the database.

Primary Key: parameterID

Foreign Key: unitID

Table 4.1-2. GeoPhysicalParms Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
parameterID	integer	NOT NULL	1	2^{16}	Unique Parameter Identifier
parameterName	varchar (32,16)	NOT NULL	N/A	N/A	Alpha-numeric name of a geophysical parameter.
minValidValue	integer	NOT NULL	MIN_FLOAT	MAX_FLOAT	Minimum valid value for parameter
maxValidValue	integer	NOT NULL	MIN_FLOAT	MAX_FLOAT	Maximum valid value for parameter
unitID	integer	NOT NULL	1	2^{16}	Identifier for default unit in which parameter is measured. Overridden by the unit actually assigned to the data item.

PRELIMINARY

4.1.3.3 Grid Data Detail Table

Table Name: GridData

Description: The Grid Data Detail Table stores detail information for a gridded data set. There will be many instantiations of this table. The table structure is shown in Table 4.1-3.

Primary Key: parameterID/datasetID/tau/level

Foreign Key: parameterID, datasetID, paramUnit, levelUnit

Table 4.1-3. gridData Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
datasetID	integer	NOT NULL	1	232	Identifier of dataset to which grid belongs
level	real	NOT NULL	MIN_FLOAT	MAX_FLOAT	Vertical Level at which parameter is modeled
levelUnit	integer	NOT NULL	0	216	Unit in which level is represented
parameterID	integer	NOT NULL	0	216	Parameter contained in grid data
paramUnit	integer	NOT NULL	0	216	Unit in which parameter is represented
tau	interval hour to minute	NOT NULL	0	240	Forecast period
theData	byte	NOT NULL	Parameter specific		Gridded data values

PRELIMINARY

4.1.3.4 Grid Dataset Directory

Table Name: GridDataSets

Description: This table is a directory of data sets. It stores summary information about each data set, which includes a reference to the bounding areas that encompass all data in the data set. The data set directory provides a quick way to determine which grids and which model from which center are in a data set, and the relevant times and bounding area, without having to query each individual grid dataset table. The table structure is shown in Table 4.1-4.

Primary Key: datasetID

Foreign Key: generatingProc, centerID, gridID

Table 4.1-4. Grid Data Set Directory Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
basetime	datetime-year-to-hour	NOT NULL	0001-01-01.00:00	9999-12-31.23:59	Basetime of model run.
centerID	smallint	NOT NULL	0	254	Numeric identifier of generating production center per WMO-306 Section C Table C-1
createTime	datetime year to fraction(5)	NOT NULL	0001-01-01.00:00	9999-12-31.23:59	Time at which dataset was created
datasetID	serial(1)	NOT NULL	0	232	Unique Identifier of a dataset.

PRELIMINARY

Table 4.1-4. Grid Data Set Directory Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
dataSetName	varchar(18)	NOT NULL	N/A	N/A	Alphanumeric name of grid dataset, also used as name of detail table that stores the grid.
generatingProc	smallint	NOT NULL	0	254	ID of generating process for model data
gridID	smallint	NOT NULL	0	254	ID of geographic area covered by model
maxTau	interval	NOT NULL	0	240	Maximum forecast period (tau) within the dataset.
retentionTime	interval	NOT NULL	00:00.0000 1	9999- 365.23:59. 99999	Number of Hours after basetime dataset should be retained.

PRELIMINARY

4.1.3.5 *Lambert/Mercator Projection Table*

Table Name: LambertMercator
Description: Contains Lambert Conformal or Mercator projection information for models.
Primary Key: gridID/centerID
Foreign Key: gridID, centerID

Table 4.1-5. LambertMercator Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
centerID	smallint	NOT NULL	0	254	ID of center that defined projection
gridID	smallint	NOT NULL	0	254	Grid ID assigned by center
standardLat1	real	NOT NULL	-90.0	90.0	Northern standard latitude
standardlat2	real	NOT NULL	-90.0	90.0	Southern standard latitude
standardLon	real	NOT NULL	-180.0	180.0	Longitude on central meridian of projection
zLambertRes	real	NOT NULL	MIN_FLO AT	MAX_FLO AT	Number of vertical levels in data

PRELIMINARY

4.1.3.6 Linear Conversion Table

Table Name: LinearConversions

Description: Stores scale and reference factors to convert data from one unit of measure to another.

Primary Key: unitID/equivalentUnit

Foreign Key: unitID, equivalentUnit

Table 4.1-6. LinearConversions Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
equivalentUnit	integer	NOT NULL	1	216	ID of unit to which unit identified by unitID will be converted
reference	real	NOT NULL	MIN_FLOAT	MAX_FLOAT	Reference value for conversion
scale	real	NOT NULL	MIN_FLOAT	MAX_FLOAT	Scale for conversion
unitID	integer	NOT NULL	1	216	ID of unit to be converted to equivalent unit

PRELIMINARY

4.1.3.7 Model Reference Table

Table Name: Models
Description: Provides definitions of geophysical models that generate grid data.
Primary Key: centerID/gridID/generatingProc
Foreign Keys: centerID, gridID

Table 4.1-7. Models Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
centerID	smallint	NOT NULL	0	254	Production center ID from WMO-306, Vol. 1, Section C, Table C-1
generatingProc	smallint	NOT NULL	0	254	Identifier of Computer process that generated the grid.
gridID	smallint	NOT NULL	0	254	Identifier for geographic area for which data are modeled
modelName	varchar(16,64)	NULL	N/A	N/A	Alphanumeric name of model
modelVersion	character(8)	NULL	N/A	N/A	Version of model

PRELIMINARY

4.1.3.8 Polar Stereographic Projection Table

Table Name: PolarStereographic
Description: Contains polar stereographic projection information for model data.
Primary Key: gridID/centerID
Foreign Key: none

Table 4.1-8. PolarStereographic Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
centerID	smallint	NOT NULL	0	254	ID of center that defined the projection
gridID	smallint	NOT NULL	0	254	Grid ID assigned by center
standardLat	float	NOT NULL	-90.0	90.0	Standard latitude for projection
standardLon	float	NOT NULL	-180.0	180.0	Standard longitude for projection
zResolution	float	NOT NULL	MIN_FLOAT	MAX_FLOAT	Vertical resolution

PRELIMINARY

4.1.3.9 Production Centers Table

Table Name: ProductionCenters

Description: Holds information about data production/transmission centers.

Primary Key: centerID

Foreign Key: none

Table 4.1-9. ProductionCenters Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
centerID	smallint	NOT NULL	0	254	Unique Numeric Identifier of Generating Production Center as identified in WMO-306 V 1 Section C table C-1.
productionCenter	varchar()	NOT NULL	N/A	N/A	Alphanumeric (ASCII) Identifier of Generating Production Center as identified in WMO-306 V 1 Section C table C-1.

PRELIMINARY

4.1.3.10 Registrations Table

Table Name: Registrations
Description: Contains information relating to the geographic registration of grids.
Primary Key: centerID/gridID
Foreign Key: centerID, gridID, aoiID

Table 4.1-10. Registrations Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
aoiID	integer	NOT NULL	1	232	Unique numeric identifier of AOI which bounds the grid
centerID	smallint	NOT NULL	0	254	Unique Numeric Identifier of Generating Production Center as identified in WMO-306 V 1 Section C table C-1.
gridID	smallint	NOT NULL	0	254	Center-assigned identification of registration.
maxXPoints	float	NOT NULL	2	10,000	Number of points along a line of longitude.
maxYPoints	integer	NOT NULL	2	10,000	Number of points along a line of latitude
maxZPoints	real	NULL	1	10,000	Number of vertical points.

PRELIMINARY

Table 4.1-10. Registrations Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
projection	integer	NOT NULL	0	29	Geographic projection of the data
registeredX	real	NOT NULL	1	maxXPoints	X Point on grid which maps to Registered Longitude
registeredY	real	NOT NULL	1	maxYPoints	Y Point on grid which maps to Registered latitude
registeredZ	real	NULL	1	maxZPoints	Z value in volume that maps to Vertical Point
regLatitude	real	NOT NULL	-90.0	90.0	Latitude of grid that maps to registeredY
regLevel	real	NULL	MIN_FLOAT	MAX_FLOAT	Vertical Point in volume that registers to Z Value
regLongitude	real	NOT NULL	-180.0	180.0	Longitude of grid that maps to registeredX
regName	varchar(32)	NOT NULL	N/A	N/A	Alpha/Numeric Name of the registration.
scanMode	integer	NOT NULL	0	7	Layout of points in a grid.

PRELIMINARY

4.1.3.11 Spherical Projection Table

Table Name: Spherical

Description: Contains spherical projection information about models.

Primary Key: gridID/centerID

Foreign Key: gridID, centerID

Table 4.1-11. Spherical Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
centerID	smallint	NOT NULL	0	254	WMO-306 ID of center that defined projection
gridID	smallint	NOT NULL	0	254	Center-assigned ID of grid
xResolution	real	NOT NULL	0.0	90.0	Grid spacing between X points measured in degrees.
yResolution	real	NOT NULL	0.0	90.0	Grid spacing between Y points measured in degrees.
zResolution	real	NOT NULL	MIN_FLOAT	MAX_FLOAT	Spacing between Z points measured in level units of data

PRELIMINARY

4.1.3.12 Units Table

Table Name:

Units

Description:

Stores identifiers for units of measure.

Primary Key:

unitID

Table 4.1-12. Units Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
unitAbrv	varchar(16)	NOT NULL	N/A	N/A	Abbreviated name of the unit
unitID	integer	NOT NULL	1	216	Numeric identifier for a unit of measure
unitName	varchar(32)	NOT NULL	N/A	N/A	Alphanumeric name of a unit

PRELIMINARY

4.2 LLT Observation Data Segment (MDLLT) Design

4.2.1 MDLLT Conceptual Design

Observational data is viewed as a class hierarchy. This class hierarchy is documented in Figure 4.2-1 through Figure 4.2-4. The Root class MDLOD-CLASS contains information that is common to all observation. All observations inherit these attributes from the root class. Each Sub-class contributes it's own attributes, which in turn, may also be inherited by it's sub-classes.

The rational behind the design is that application requirements for observation data is varied and observation data is received from specific meteorological formatted messages. The best way to accommodate application requirements is to provide different views into the database. The best way to accommodate ingestion of messages is to provide structures based on groupings of fields within messages. These views, in the logical model, are established by the class hierarchy. Leaf level classes support message level structures, while higher level classes support common views of observations fields that are more useful to applications that aren't concerned with what message a particular datum was transmitted in. Or want to look at conditions that are within several different messages.

The physical observation data model maps the class hierarchy on to a relational data model. The same structural requirements exist as were noted in the logical model, however several pragmatic issues must be addressed with the physical model. The first issue is how to map an object model onto a relational model. The second is managing the perishability of the data. This involves purging out data when it is no longer useful as well as managing the sizes of tables so they do not become unruly. Lastly, providing logical groupings of data with regard to space and time so as to optimize the storage and retrieval of data.

PRELIMINARY

The first issue, mapping an object oriented design onto a relational database was solved using techniques outlined in the paper “Object-Oriented Technology for Integrating Distributed Heterogeneous Database Systems”¹. The techniques outlined in the paper suggest “object identifiers of the instances can be used as primary keys ... Object classes can be arranged into a class hierarchy.” A “Vehicle class has Ship as it’s subclass. Similarly, SHIP has SURFACE SHIP and SUBMARINE as object subclasses. A domain exists for each class and subclass. Depending on the degree of normalization, the relation variable generated for the subclass may contain only the attributes that distinguish the subclass from other subclasses or it could include additional attributes also found in the superclass.”².

In order to implement this, three classes are defined:

1. Surface Obs
2. Ocean Obs
3. Atmospheric Obs.

These classes are derived from a common parent class: MDLOD-CLASS. MDLOD-CLASS domain is SURFACE-OB, OCEAN-OB and ATMOSPHERIC-OB. OCEAN-OB’s domain is BUOY-CLASS and BATHY-CLASS. SURFACE-OB-CLASS and ATMOSPHERIC-OB-CLASS have domains as well.

Each of the classes may have “container” classes. That is, class attributes that are instances of other classes. An example is the attribute LLTOB-LOCATION with in MDLOD-CLASS. This attribute is defined as a LLT-LOCATION-CLASS. The domain of LLT-LOCATION-CLASS is LLT-SHIP-LOCATIONS, LLT-BATHY-LOCATIONS, LLT-LAND-STATION-LOCATIONS and AIRCRAFT-

¹ Object Oriented Technology For Integrated Distributed Heterogeneous Database System, Dr. Marion Ceruti, Dr Magdi N. Kamel, Dr. Bhavani M. Thuraisingham.

² Ibid.

PRELIMINARY

LOCATIONS. The particular sub-class is dependent on the MDLOD-CLASS or sub-class.

Tables are defined with the primary key being an object-id. Alternate Keys are defined with attributes that would otherwise be the primary key. Instantiation of an object will be done through the use of queries within the API. Message level queries would be more complex since spanning several tables.

4.2.2 MDLLT Logical Level Design

The entity-relationship diagrams on the pages that follow depict the logical model of the LLT Observation Data Segment. Separate entity relationship diagrams are provided for each observation type; note, however, that all types share a common root.

PRELIMINARY

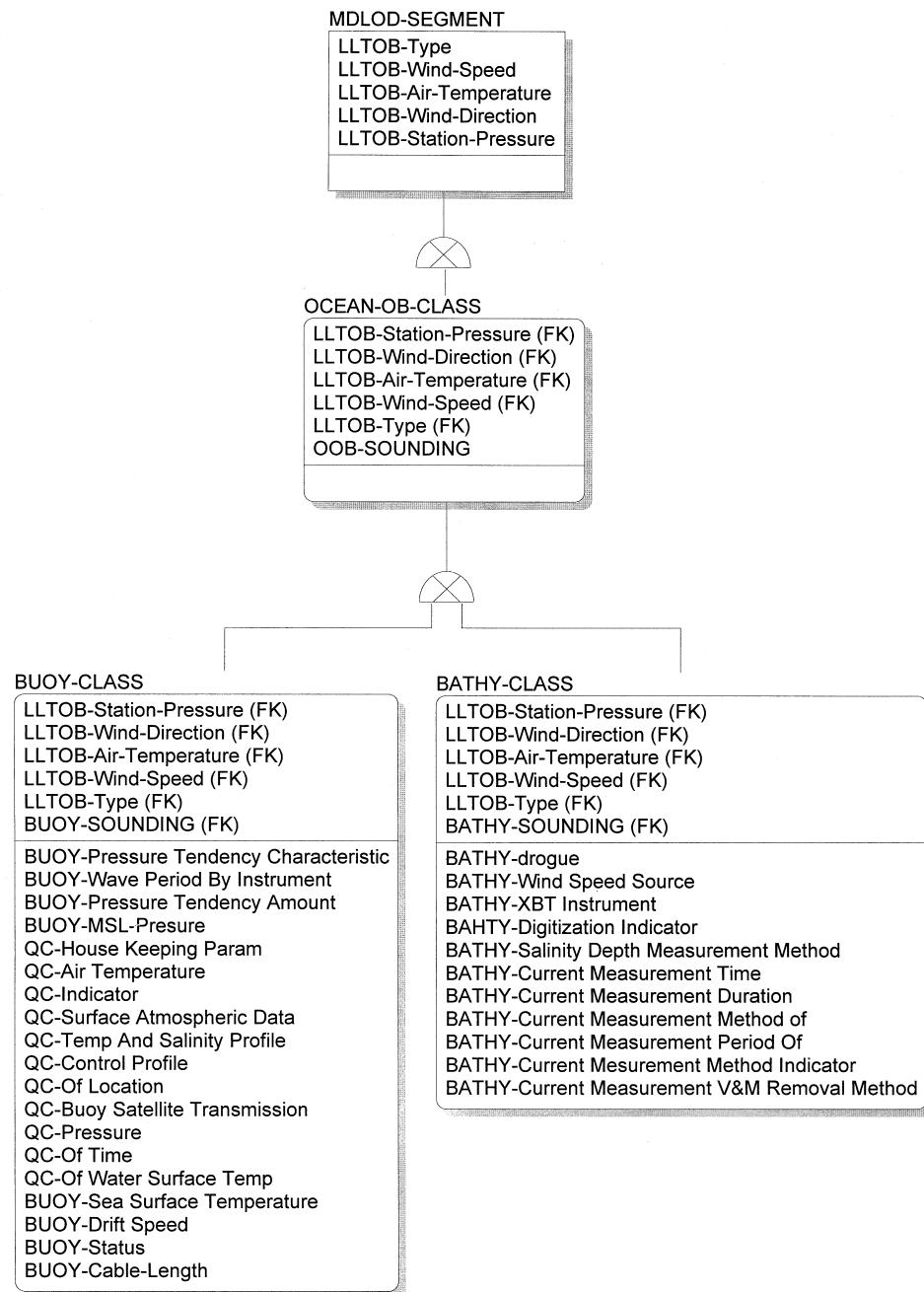


Figure 4.2-1. Ocean Observation Data Logical Model

PRELIMINARY

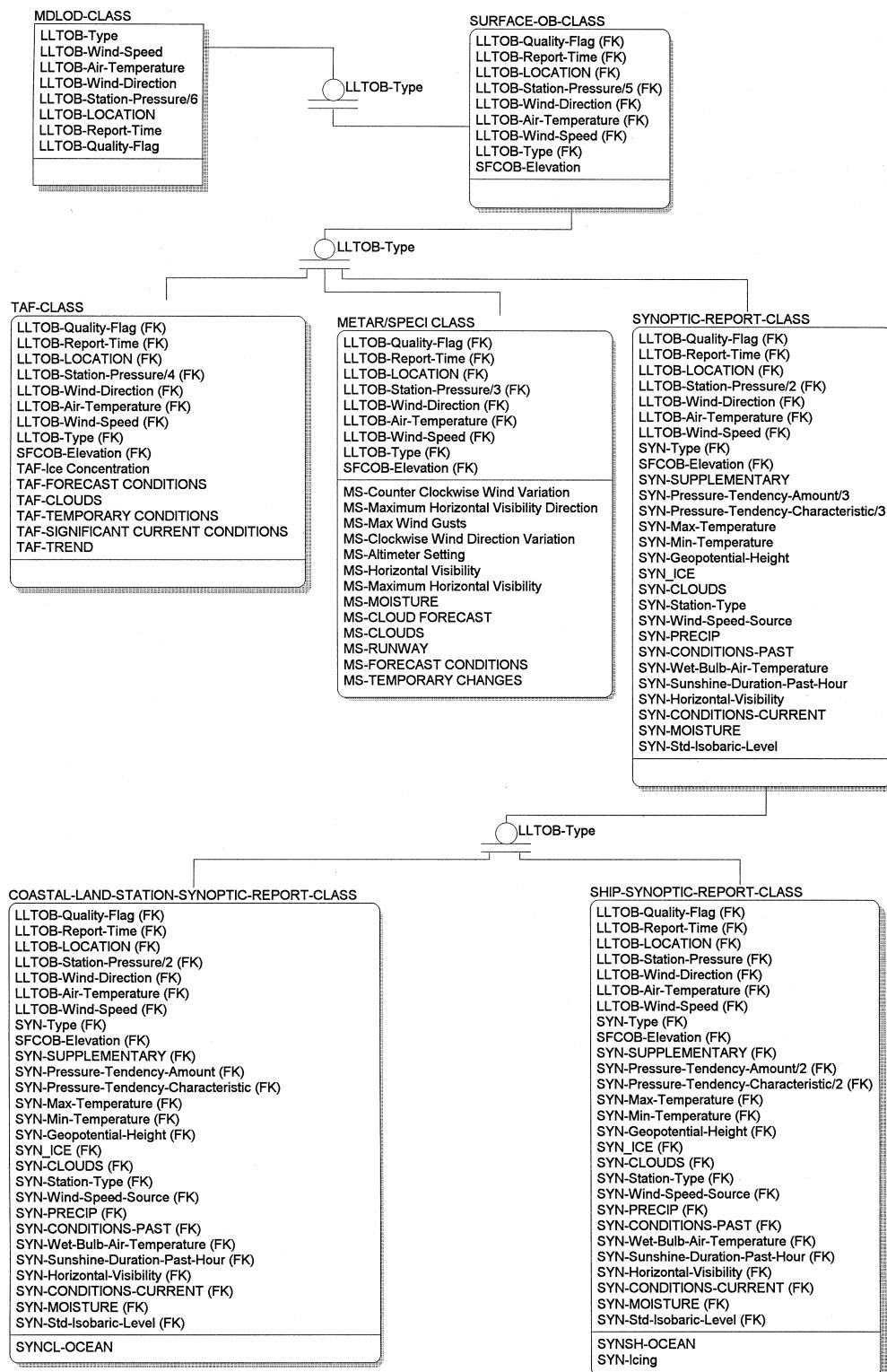


Figure 4.2-2. Surface Observation Data Logical Model

PRELIMINARY

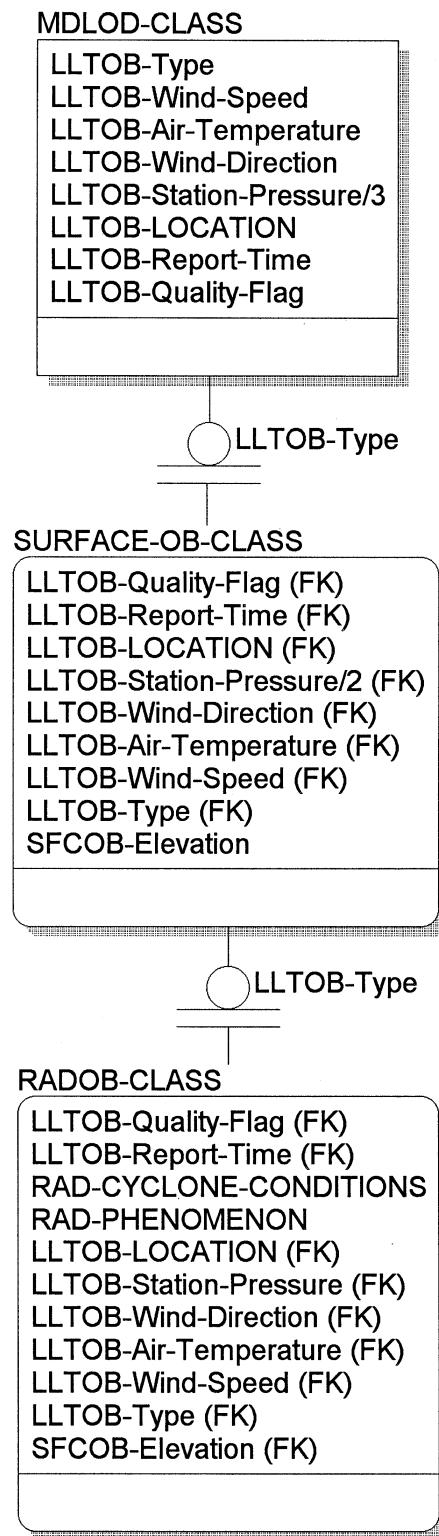


Figure 4.2-3. Radar Observation Data Logical Model

PRELIMINARY

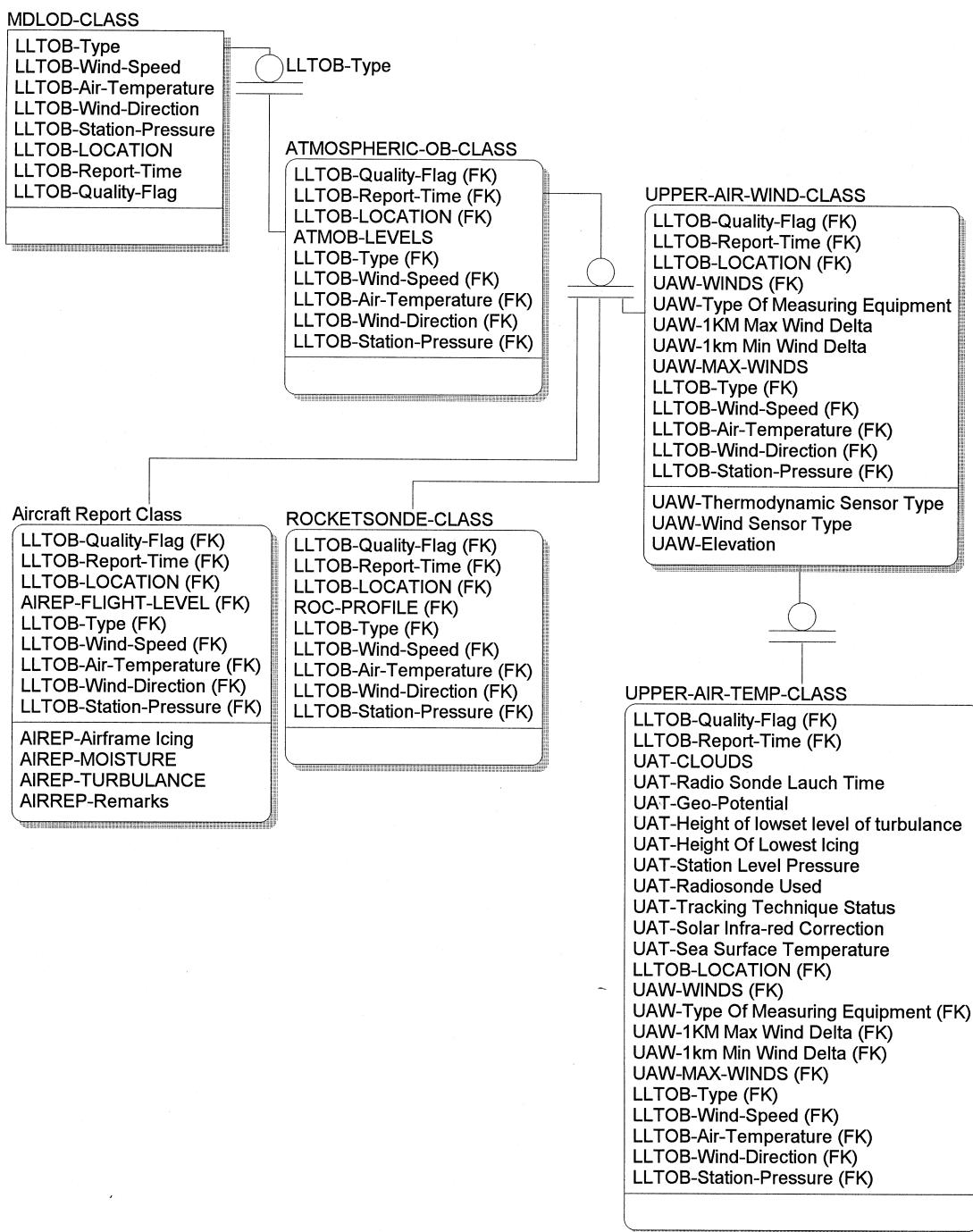


Figure 4.2-4. Upper Air Observation Data Logical Model (Part 1 of 2)

PRELIMINARY

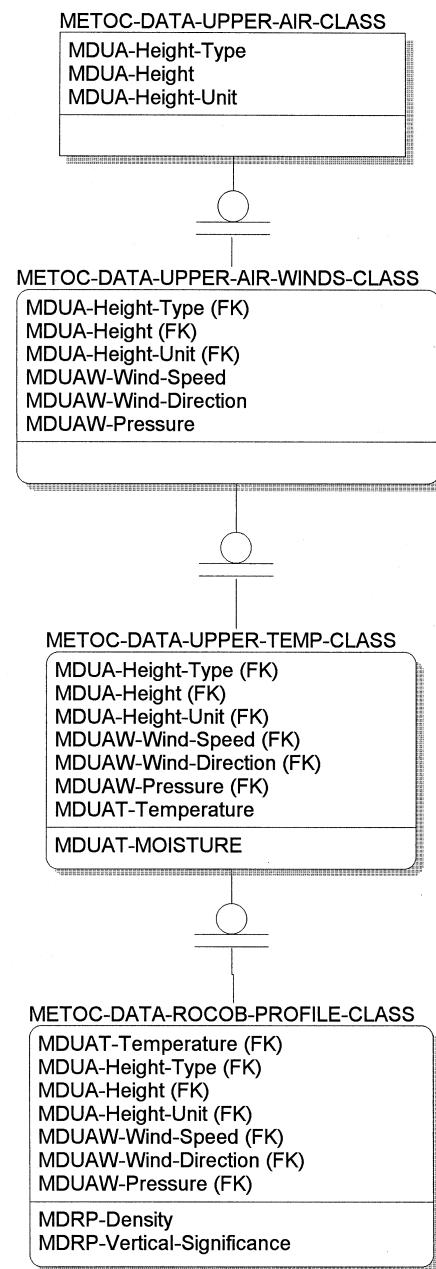


Figure 4.2-4. Upper Air Observation Data Logical Model (Part 2 of 2)

PRELIMINARY

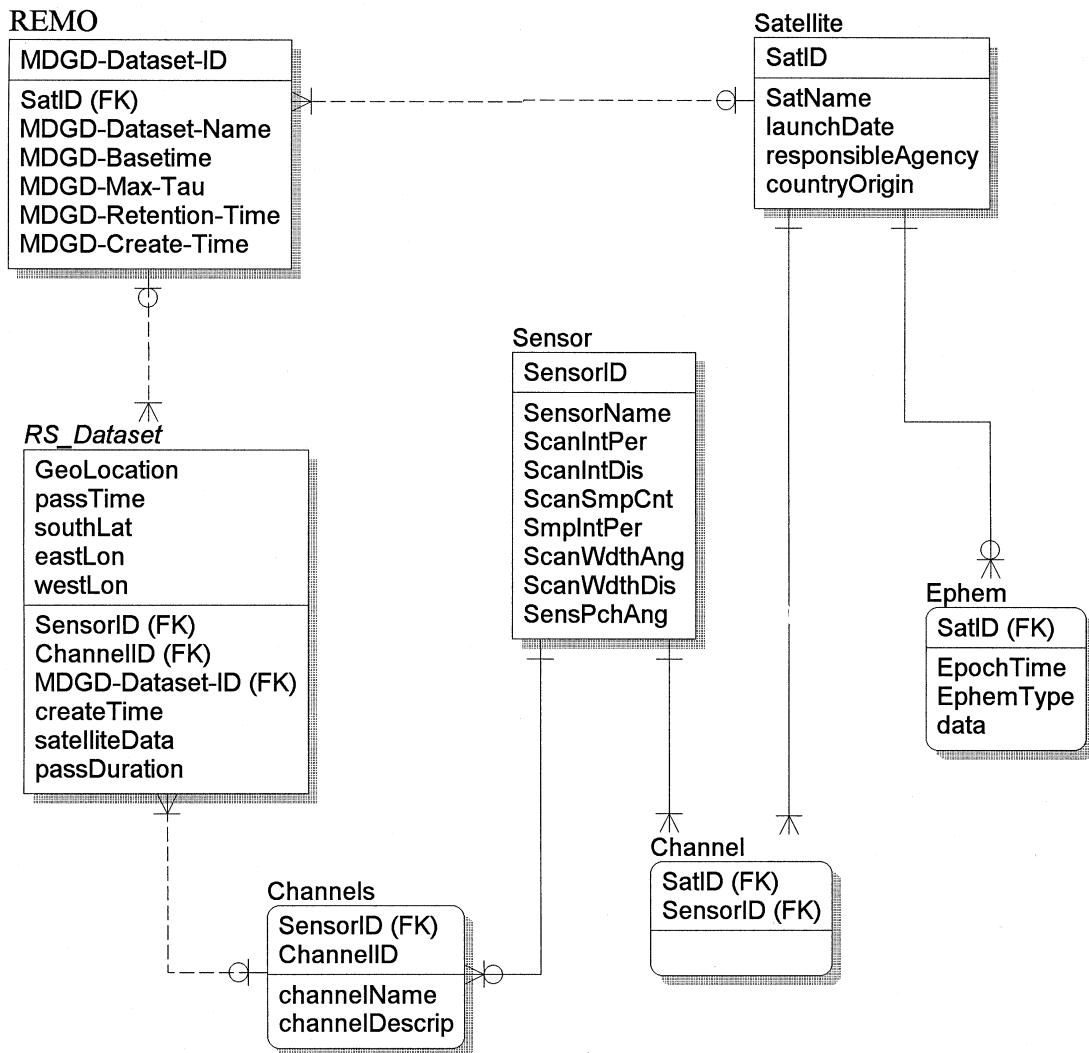


Figure 4.2-5. Remotely Sensed Data Logical Model

PRELIMINARY

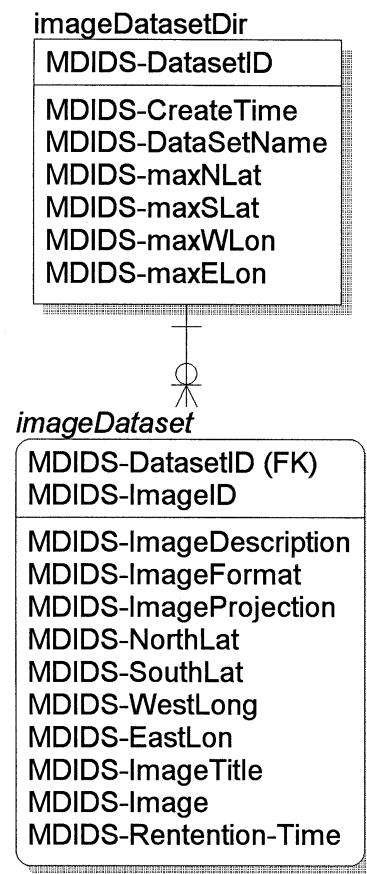


Figure 4.2-6. Logical Model for Image Data

PRELIMINARY

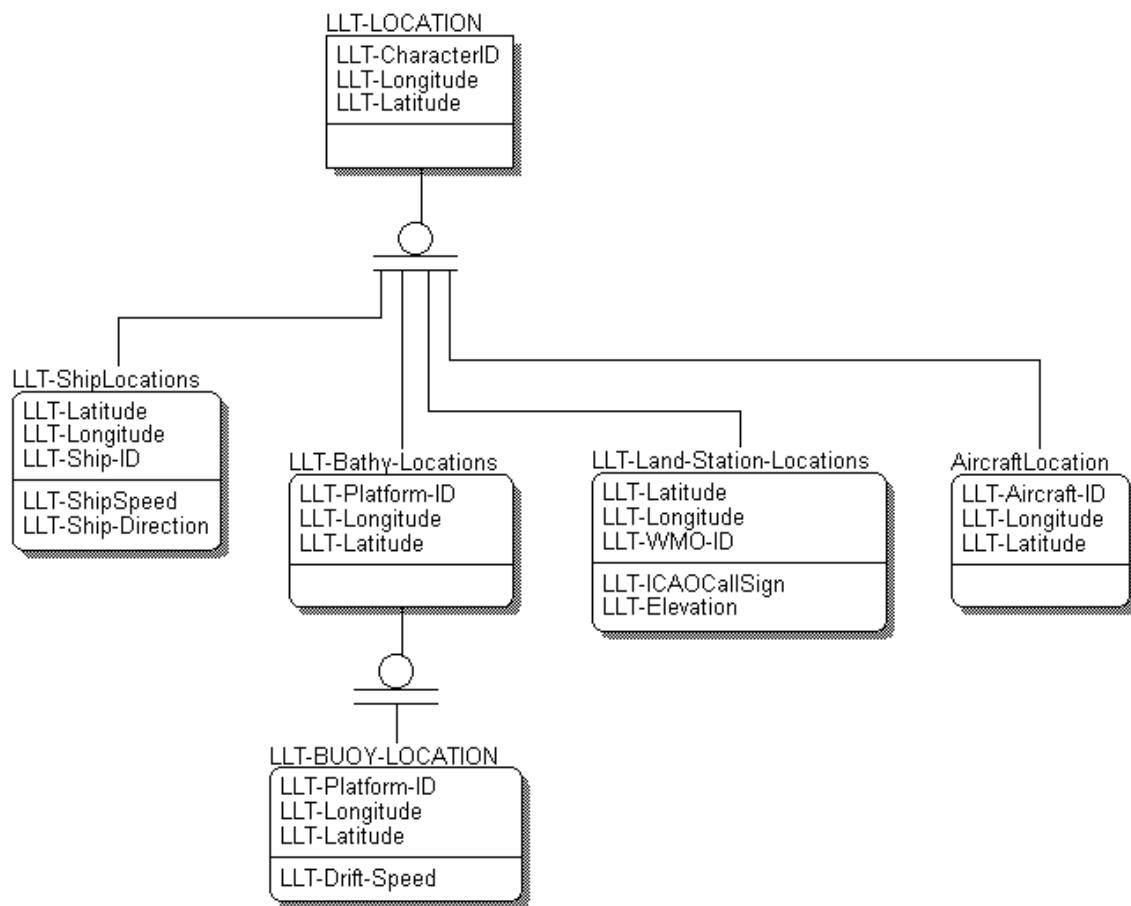


Figure 4.2-7. Logical Model for LLT Data Location Information

PRELIMINARY

4.2.3 MDLLT Physical Level Design

This section presents the designs of the individual tables that make up the MDLLT. The design is broken out by type of observation. The objective of the physical design of the database is to optimize access to data and to take advantage of RDBMS capabilities. In order to do this, like parameters have been grouped together. Section 4.2.3.1 discusses database tables common to all observation types; the remaining subsections describe the physical level design for each observation type separately.

4.2.3.1 Database Tables Common to All Observation Types

4.2.3.1.1 Observation Areas of Interest Table

Table Name: obAOIs

Description: Stores information about area of interest boundaries for observations.

Primary Key: aoiID

Table 4.2-1. obAOIs Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
aoiID	serial(1)	NOT NULL	1	232	Unique numeric identifier for the AOI.
aoiName	varchar(64)	NOT NULL	N/A	N/A	Alphanumeric name of AOI.
eastLon	smallfloat	NOT NULL	-180.0000000	180.0000000	Easternmost bounding vector of AOI.
northLat	real	NOT NULL	-90.0000000	90.0000000	Northernmost bounding vector of AOI.
southLat	real	NOT NULL	-90.0000000	90.0000000	Southernmost bounding vector of AOI.
westLon	real	NOT NULL	-180.0000000	180.0000000	Westernmost bounding vector of AOI.

PRELIMINARY

4.2.3.1.2 Observation Collection Areas Table

Table Name: obCollectionAreas

Description: Contains information about the areas and times over which groups of observations were collected.

Primary Key: obTypeID/obSubType

Foreign Key: aoiID

Table 4.2-2. obCollectionAreas Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
aoiID	integer	NOT NULL	1	232	Unique identifier for AOI
maxObs	integer	NOT NULL	1	232	Maximum number of observations to ingest into dataset.
maxOpenTime	interval	NOT NULL	0:01	23:59	Maximum time for which a dataset will be opened for ingestion of data.
obSubType	smallint	NOT NULL	1	200	Observation data subtype
obTypeID	smallint	NOT NULL	1	65535	Type of observation into which data is to be ingested
purgeIncr	interval hour (9) to minute	NOT NULL	1	999999999	Increment, from time dataset was created, that data set will be purged.

PRELIMINARY

4.2.3.1.3 Observation Dataset Directory Table

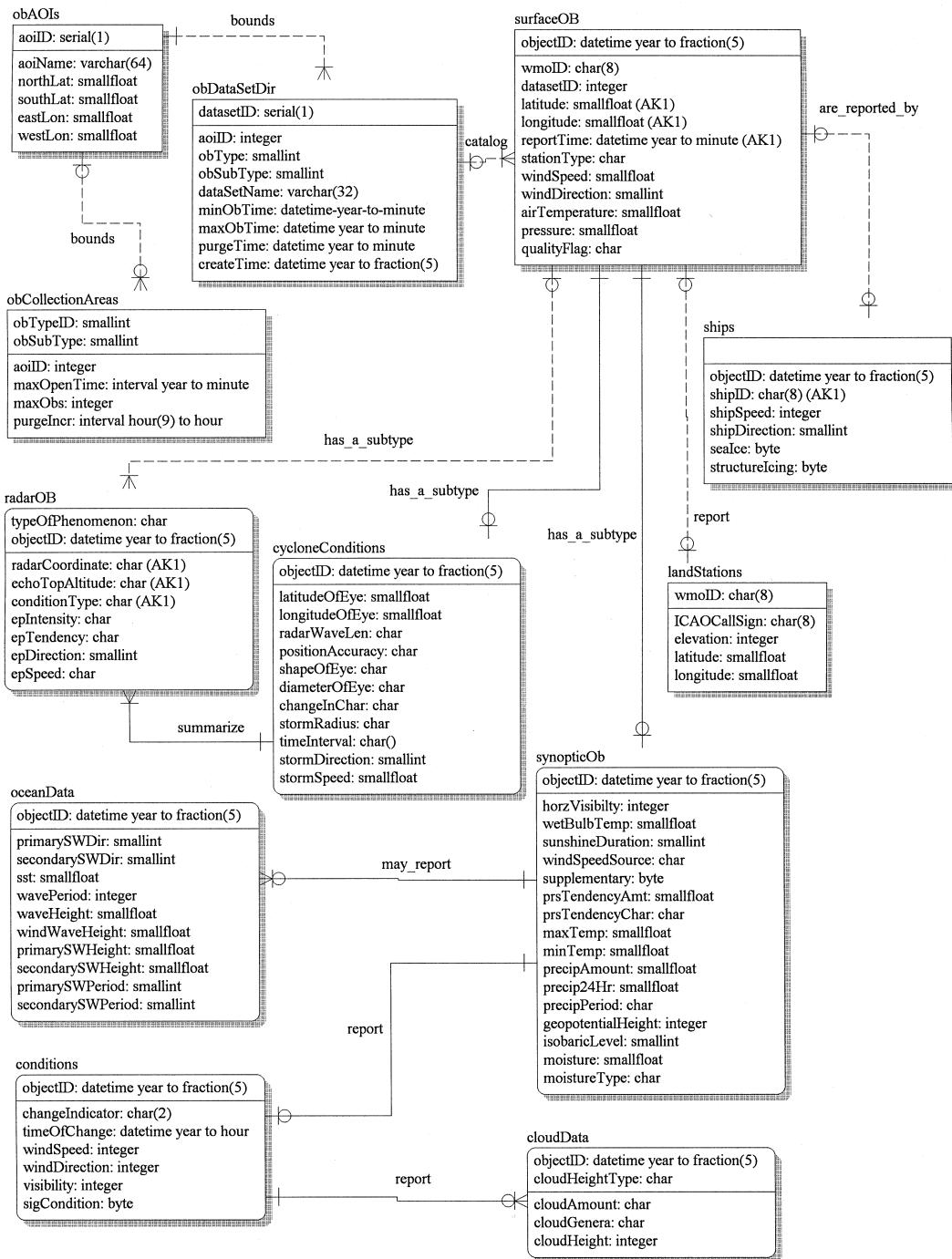
Table Name: obDataSetDir
Description: Stores information about an observation dataset.
Primary Key: datasetID
Foreign Key: aoiID

Table 4.2-3. ObDataSetDir Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
aoiID	integer	NOT NULL	1	232	ID of AOI used as the bounding area of the dataset.
createTime	datetime year to fraction(5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Time at which dataset was created.
datasetID	serial(1)	NOT NULL	0	232	Unique Identifier of a dataset.
dataSetName	varchar(32)	NOT NULL	N/A	N/A	Alphanumeric name of Ob dataset, also used as name of detail table which stores the grid.
maxObTime	datetime year to minute	NOT NULL	'0001 01-01 00:00'	'9999 12-31 23:59'	Maximum report time of an observation in dataset.
minObTime	datetime-year-to-minute	NULL	'0001 01-01 00:00'	'9999 12-31 23:59'	Minimum report time of an observation in dataset.
obSubType	smallint	NOT NULL	1	200	Observation subtype stored in dataset.
obType	smallint	NOT NULL	1	65535	Type of observation being stored in the dataset.
purgeTime	datetime hour to minute	NOT NULL	0001-01-01 00:00	9999-12-31 23:59	Time at which to purge the dataset.

PRELIMINARY**4.2.3.2 Physical Level Design for Surface Observations**

Figure 4.2-8 presents the physical level design for surface observations as entity-relationship diagrams.

**Figure 4.2-8. Physical Level Design for Surface Observations**

PRELIMINARY

The remainder of this section presents the detailed design of the tables used for surface observations.

4.2.3.2.1 Cloud Data Table

Table Name:	cloudData
Description:	Contains information concerning amount, height, and type of clouds reported.
Primary Key:	objectID/cloudHeightType
Foreign Key:	objectID

Table 4.2-4. cloudData Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
cloudAmount	char	NULL	0	9	Amount of Clouds in sky.
cloudGenera	char	NULL	0	9, '/'	Genera of cloud.
cloudHeight	integer	NULL	0	99	Altitude of upper surface of clouds whose base is below the level of the station, in hundreds of meters
cloudHeightType	character	NOT NULL	0	9	Type of cloud: low, middle, or high
objectID	datetime year to fraction(5)	NOT NULL	0001-01-01 00:00:00.00 000	9999-12-31 23:59:59.99 999	Unique identifier for the object; also its creation time

PRELIMINARY

4.2.3.2.2 Conditions Table

Table Name: conditions
Description: Contains data concerning current and expected weather conditions.
Primary Key: objectID
Foreign Key: objectID

Table 4.2-5. conditions Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
changeIndicator	char(8)	NULL	N/A	N/A	Indicator of weather change FROM, AT, UNTIL, TEMPO, BECMG
objectID	datetime year to fraction(5)	NOT NULL	0001-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique identifier of the object; also its creation time
sigCondition	byte	NULL	N/A	N/A	Text of significant condition indicator
timeOfChange	datetime year to hour	NULL	0001-01-01 00:00	9999-12-31 23:59	Time associated with change-Indicator.
visibility	integer	NULL	000000	160000	Prevailing Visibility.
windDirection	integer	NULL	0.0	359.0	Wind Direction of condition
windSpeed	integer	NULL	0.0	200.00	Wind Speed of condition

PRELIMINARY

4.2.3.2.3 Cyclone Conditions Table

Table Name: cycloneConditions
Description: Contains information about tropical cyclones based on radar reports.
Primary Key: objectID
Foreign Key: objectID

Table 4.2-6. cycloneConditions Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
changeInChar	char	NULL	0	9, '/'	Change in character of eye during previous 30 minutes preceding the time of observation.
diameterOfEye	char	NULL	0	9, '/'	Diameter or length of major axis of the tropical cyclone.
latitudeOfEye	smallfloat	NULL	-90.0000000	90.0000000	Latitude of eye of tropical storm.
longitude OfEye	smallfloat	NULL	-90.0000000	90.0000000	Longitude of eye of tropical cyclone.
objectID	datetime year to fraction(5)	NOT NULL	0001-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique identifier for object; also its creation time
position Accuracy	char	NULL	1	7, '/'	Accuracy of position of the center of the storm.
radarWaveLen	char	NULL	1,3,5,7,9		Code for wave length of radar. Code table 3555

PRELIMINARY

Table 4.2-6. cycloneConditions Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
shapeOfEye	char	NULL	0	5, '/'	Shape and definition of eye of hurricane.
stormDirection	integer	NULL	0	359	Direction towards which storm is moving.
stormRadius	char	NULL	0	9, '/'	Distance between the end of the observed spiral band and the center of the tropical cyclone.
stormSpeed	smallfloat	NULL	0.0	300.00	Forward speed at which storm is moving.
timeInterval	integer	NULL	9	9, '/'	Period of time over which movement of storm center has been calculated.

PRELIMINARY

4.2.3.2.4 Land Stations Table

Table Name: landStations

Description: Contains identifier and elevation information for land stations.

Primary Key: wmoID

Table 4.2-7. landStations Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
elevation	integer	NULL	-999	99999	Height above sea level of station in meters
ICAO CallSign	char(8)	NULL			International Civil Aviation Organization assigned call sign.
latitude	smallfloat	NULL	-90.0000000	90.0000000	Latitude of station
longitude	smallfloat	NULL	-180.0000000	180.0000000	Longitude of station
wmoID	char(8)	NOT NULL	'00000'	'89999'	WMO Block Station Number designating geographic location of report.

PRELIMINARY

4.2.3.2.5 Ocean Data Table

Table Name: oceanData

Description: Stores ocean conditions data from surface observations.

Primary Key: objectID

Foreign Key: objectID

Table 4.2-8. oceanData Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
objectID	datetime year to fraction(5)	NOT NULL	0001-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Object identifier
primary SWDir	smallint	NULL	0	359	The angle measured clockwise from true north to the direction from which primary swell waves are coming.
primary SWHeight	smallfloat	NULL	0.00	50.00	Height of the primary swell waves, from trough to crest
primary SWPeriod	smallint	NULL	0	14	Time elapsed between passage of successive primary swell wave crests

PRELIMINARY**Table 4.2-8. oceanData Table Structure**

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
secondary SWDir	smallint	NULL	0.0	359.0	The angle measured clockwise from true north to the direction from which secondary swell waves are coming.
secondary SWHeight	smallfloat	NULL	0.00	50.00	Height of secondary swell waves, from trough to crest
secondary SWPeriod	smallint	NULL	0	14	Time elapsed between passage of successive crests of secondary swell
sst	smallfloat	NULL	-110.0	63.0	Sea-surface temperature.
waveHeight	smallfloat	NULL	0.00	50.00	Mean height of highest 1/3 of waves
wavePeriod	integer	NULL	0	14	Period of wind-driven waves
windWave Height	smallfloat	NULL	0.00	50.00	Height of wind-driven waves

PRELIMINARY

4.2.3.2.6 Radar Observation Table

Table Name: radarOB**Description:** Contains information about phenomena observed by radar.**Primary Key:** typeOfPhenomenon/objectID**Foreign Key:** objectID**Alternate Keys:** radarCoordinate, echoTopAltitude, conditionType**Table 4.2-9. radarOb Table Structure**

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
conditionType	char	NULL	1	9, '/'	Type of weather phenomenon or cloud in the 60km, 60km square detected by radar.
echoTop Altitude	character	NULL	0	9, '/'	Height above ground of echo
epDirection	char	NULL	0	9, '/'	Direction in which echo is moving.
epIntensity	char	NULL	0	9, '/'	Intensity of echo pattern described as weak to very strong.
epSpeed	char	NULL	0	9, '/'	Speed at which echo is moving.
epTendency	char	NULL	1	9, '/'	Change in intensity of and coverage of echo, expressed as increasing or decreasing or no appreciable change.

PRELIMINARY

Table 4.2-9. radarOb Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
objectID	datetime year to fraction(5)	NOT NULL	0001-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique identifier for object
radar Coordinate	char	NULL	00	99	Sequence number in the 60X60 km radar coordinate grid
typeOf Phenomenon	char	NOT NULL	1	9, '/'	Type of phenomenon observed by radar

PRELIMINARY

4.2.3.2.7 Ships Table

Table Name:

ships

Description:

Contains ship ID, speed, and direction information.

Foreign Key:

objectID

Alternate Key:

shipID

Table 4.2-10. ships Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
objectID	datetime year to fraction(5)	NOT NULL	0001-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique identifier for object
seaIce	byte	NULL	0	N/A	Sea ice data reported
shipDirection	smallint	NULL	0	359	Direction in which ship is traveling
shipID	char(8)	NOT NULL	N/A	N/A	Identification of the ship.
shipSpeed	integer	NULL	0	50	Speed at which ship is traveling.
structureIcing	byte	NULL	N/A	N/A	Structure icing data reported

PRELIMINARY

4.2.3.2.8 Surface Observation Table

Table Name: surfaceOB**Description:** Stores information from a single surface observation.**Primary Key:** objectID**Foreign Key:** wmoID, datasetID**Alternate Keys:** latitude, longitude, reportTime**Table 4.2-11. surfaceOB Table Structure**

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
airTemperature	smallfloat	NULL	-110.0	63.0	Air temperature at reporting station.
datasetID	integer	NULL	0	232	Unique Identifier of a dataset.
latitude	smallfloat	NOT NULL	-90.0000000	90.0000000	Latitude of reporting station.
longitude	smallfloat	NOT NULL	-180.0000000	180.0000000	Longitude of reporting station.
objectID	datetime year to fraction(5)	NOT NULL	0001-01-01 00:00:00.000 00	9999-12-31 23:59:59.999 99	Unique identifier for object; also its creation time
pressure	smallfloat	NULL	450.00	1100.00	Pressure at station level.
qualityFlag	char	NULL	N/A	N/A	Flag for quality of observation
reportTime	datetime year to minute	NOT NULL	0001-01-01 00:00	9999-12-31 23:59	Time at which observation was made.
stationType	char	NULL	1	7	Type of station operation
windDirection	smallint	NULL	0	359	Wind direction at reporting station.

PRELIMINARY

Table 4.2-11. surfaceOB Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
windSpeed	smallfloat	NULL	0.0	200.0	Wind Speed at reporting station.
wmoID	char(8)	NULL	'00000'	'89999'	WMO Block Station Identifier

PRELIMINARY

4.2.3.2.9 Synoptic Observation Table

Table Name: synopticOb**Description:** Stores fields from a synoptic observation.**Primary Key:** objectID**Table 4.2-12. synopticOb Table Structure**

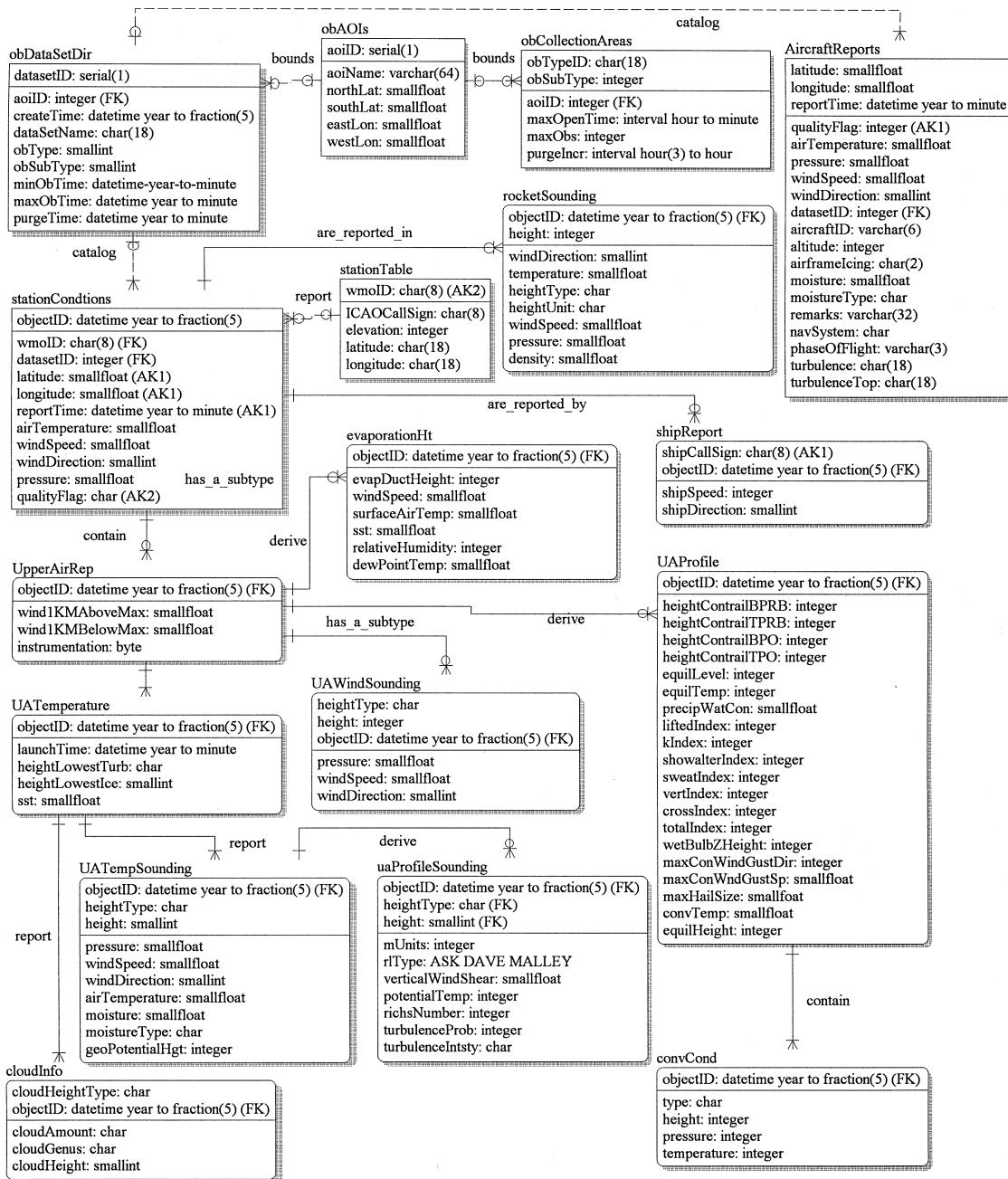
Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
geopotential Height	integer	NULL	0	150000	Geopotential height at assigned standard isobaric level
horzVisibility	integer	NOT NULL	000000	160000	Horizontal visibility from surface
isobaricLevel	smallint	NULL	0	65535	A selected isobaric level
maxTemp	smallfloat	NULL	-110.0	63.0	Maximum air temperature at station over a locally defined period of time.
minTemp	smallfloat	NULL	-110.0	63.0	Minimum air temperature at station over a locally defined period of time.
moisture	smallfloat	NULL	-110.0 (dew pt) 0 (RH)	(dew pt) 100 (RH)	Moisture at flight level type of moisture is indicated by moisture type.
moistureType	char	NULL	0	1	Type of moisture reported, relative humidity (0), or dewpoint temperature (1)
objectID	datetime year to fraction(5)	NOT NULL	0001-01-01 00:00:00.0000	9999-12-31 23:59:59.9999	

PRELIMINARY**Table 4.2-12. synopticOb Table Structure**

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
precip24Hr	smallfloat	NULL	0.0	999.9	Water-equivalent precipitation amount measured over the previous 24 hours.
precipAmount	smallfloat	NULL	0.0	999.9	Amount of precipitation at station during time period preceding observation.
precipPeriod	char	NULL	000	999	Period preceding time of observation during which precipitation was accumulated
prsTendency Amt	smallfloat	NOT NULL	-80.0	80.0	Amount of pressure change at given station for the past 3 hours.
prsTendency Char	char	NULL	0	8	Characteristic of pressure tendency over past three hours.
sunshine Duration	smallint	NULL	0	5	Duration of sunshine over past hour
supplementary	byte	NULL	N/A	N/A	Gives any additional information about phenomena occurring at the time of observation.
wetBulbTemp	smallfloat	NULL	-110.0	63.0	Wet bulb air temperature.
windSpeed Source	char	NULL	0	4	Indicator of source of wind speed measure-ment

PRELIMINARY**4.2.3.3 Physical Level Design for Upper Air Observations**

Figure 4.2-9 presents the physical level design for upper air observations as entity-relationship diagrams.

**Figure 4.2-9. Physical Level Design for Storage of Upper Air Observations**

PRELIMINARY

The remainder of this section presents the detailed design of the tables used for surface observations.

4.2.3.3.1 Aircraft Reports Table

Table Name: AircraftReports
Description: Stores data elements common to aircraft reports.
Primary Key: latitude/longitude/reportTime
Foreign Key: datasetID
Alternate Key: qualityFlag

Table 4.2-13. AircraftReports Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
aircraftID	varchar(6)	NULL	N/A	N/A	Identifier of aircraft making the report.
airframeIcing	char(2)	NULL			Amount of icing on aircraft
airTemperature	smallfloat	NULL	-110.0	63.0	Air temperature at flight level.
altitude	integer	NOT NULL	0	500	Aircraft altitude in hundreds of feet
datasetID	integer	NOT NULL	1	MAXINT	A unique dataset identifier.
latitude	smallfloat	NULL	-90.0000000	90.0000000	Latitude of aircraft.
longitude	smallfloat	NULL	-180.0000000	180.0000000	Longitude of aircraft.
moisture	smallfloat	NULL	-110.0 (dew pt) 0 (RH)	(dew pt) 100 (RH)	Moisture at flight level type of moisture is indicated by moisture type.
moistureType	char	NULL	0	1	Type of moisture reported, relative humidity (0), or dewpoint temperature (1)
navSystem	char	NULL	0 (inertial)	1 (Omega)	Type of navigation system on board aircraft.

PRELIMINARY

Table 4.2-13. AircraftReports Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
phaseOfFlight	varchar(3)	NULL	LVR (routine observation in level flight), LVW (highest wind encountered in level flight), ASC (observation during ascent), DES (observation during descent), or none if unsteady flight mode.		Phase of flight aircraft is in.
pressure	smallfloat	NULL	.001	1100.000	Pressure at flight level
remarks	varchar (32)	NULL	N/A	N/A	Free form text remarks that may have been provided with report.
reportTime	datetime year to minute	NULL	0001-01-01 00:00	9999-12-31 23:59	Time at which report was made.
turbulence	char(18)	NULL	N/A	N/A	Turbulence type reported
turbulenceTop	char(18)	NULL	N/A	N/A	Highest altitude of turbulence reported
windDirection	smallint	NULL	0	359	Wind direction at flight level.
windSpeed	smallfloat	NULL	0.0	300.0	Wind Speed At flight level.

PRELIMINARY

4.2.3.3.2 Cloud Information Table

Table Name: cloudInfo
Description: Stores information concerning cloud amount and genus observed.
Primary Key: cloudHeightType/objectID
Foreign Key: objectID

Table 4.2-14. cloudInfo Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
cloud Amount	char	NULL	0	9, '/'	Amount of Clouds in sky (oktas)
cloudGenus	char	NULL	0	9, '/'	Detailed information on the selected cloud genus.
cloudHeight	smallint	NULL	0	35000	Height of cloud.
cloudHeight Type	char	NOT NULL	0	9, '/'	Type of cloud: low, medium or high. (Enumerated 0, 1, 2, respectively.)
objectID	datetime year to fraction(5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique record identifier and time stamp when record was inserted into the table.

PRELIMINARY

4.2.3.3.3 convCond Table

Table Name: convCond

Description: Holds additional station conditions information.

Primary Key: objectID

Foreign Key: objectID

Table 4.2-15. convCond Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
height	integer	NULL	0	150000	Station elevation
objectID	datetime year to fraction(5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique Identifier for record, also marks time at which record was inserted or updated.
pressure	integer	NULL	0	1100	Pressure
temperature	integer	NULL	-110	63	Temperature
type	char	NULL			Type of station

PRELIMINARY

4.2.3.3.4 Evaporation Duct Height Table

Table Name: evaporationHt

Description: Stores evaporation duct height and information used to calculate it.

Primary Key: objectID

Foreign Key: objectID

Table 4.2-16. evaporationHt Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
dewPoint Temp	smallfloat	NULL	-110.0	63.0	Dew point temperature
evapDuct Height	integer	NULL	0	150000	Evaporation duct height
objectID	datetime year to fraction(5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique Identifier for record, also marks time at which record was inserted or updated.
relative Humidity	integer	NULL	0	100	Ratio of water vapor pressure to saturation vapor pressure
sst	smallfloat	NULL	-5.0	45.0	Sea surface temperature
surfaceAir Temp	smallfloat	NULL	-110.0	63.0	Surface air temperature
windSpeed	smallfloat	NULL	0.0	300.0	Wind speed

PRELIMINARY

4.2.3.3.5 Rocket Sounding Table

Table Name: rocketSounding
Description: Stores data generated by a rocket sounding.
Primary Key: objectID/height
Foreign Key: objectID

Table 4.2-17. rocketSounding Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
density	smallfloat	NULL	.000001	1.000000	Density at given height.
height	integer	NOT NULL	0	150000	Height at which data are observed; value (dependent on heightType); has a maximum range value large enough to accommodate geopotential meters or hectopascals ranges.
heightType	char	NULL	'11'	'66'	Indicator of units in which pressure and height are coded
heightUnit	char	NULL	0	2	Unit of height measurement can be meters, hectopascals, or geopotential meters. (Enumerated as follows: 0 = m, 1 = hP, 2 = gpm)

PRELIMINARY

Table 4.2-17. rocketSounding Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
objectID	datetime year to fraction(5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique record identifier and time stamp when record was inserted into the table.
pressure	smallfloat	NULL	.001	1100.000	Pressure at given height.
temperature	smallfloat	NULL	-110.0	63.0	Temperature at given height
windDirection	smallint	NULL	0	359	Wind direction at given height.
windSpeed	smallfloat	NULL	0.0	300.0	Wind speed at given height.

PRELIMINARY

4.2.3.3.6 Ship Report Table

Table Name: shipReport

Description: Stores information common to ship reports.

Primary Key: shipCallSign/objectID

Foreign Key: objectID

Alternate Key: shipCallSign

Table 4.2-18. shipReport Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
objectID	datetime year to fraction(5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique Identifier for record, also marks time at which record was inserted or updated.
shipCallSign	char(8)	NOT NULL	N/A	N/A	Ship's international call sign; given for sea stations only
shipDirection	smallint	NULL	0	359	Bearing from true north of ship's heading
shipSpeed	integer	NULL	0	99999	Speed of the ship (in feet per sec).

PRELIMINARY

4.2.3.3.7 Station Conditions Table

Table Name: stationConditions
Description: Stores information concerning conditions at the observation station.
Primary Key: objectID
Foreign Key: datasetID, wmoID
Alternate Keys: latitude, longitude, reportTime, qualityFlag

Table 4.2-19. stationConditions Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
airTemperature	smallfloat	NULL	-110.0	63.0	Air temperature at station level.
datasetID	integer	NOT NULL	1	232	Unique identifier of the dataset.
latitude	smallfloat	NOT NULL	-90.0000000	90.0000000	Latitude of launch site.
longitude	smallfloat	NOT NULL	-180.0000000	180.0000000	Longitude of launch site.
objectID	datetime year to fraction(5)	NOT NULL	0-01-01 00:00:00.000 00	9999-12-31 23:59:59.99999	Unique record identifier; also record creation/ modification time
pressure	smallfloat	NULL	450.00	1100.00	Pressure at station level.
qualityFlag	char	NULL	N/A	N/A	Flag for data received with errors
reportTime	datetime year to minute	NOT NULL	0-01-01 00:00	9999-12-31 23:59	Time that the report was created.
windDirection	smallint	NULL	0	359	Wind direction at station level.

PRELIMINARY

Table 4.2-19. stationConditions Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
windSpeed	smallfloat	NULL	0.0	300.0	Speed of wind at station level.
wmoID	char(8)	NULL	00000	89999	WMO Block Station number

PRELIMINARY

4.2.3.3.8 Station Table

Table Name: stationTable

Description: Stores information about a reporting station.

Primary Key: wmoID

Alternate Key: wmoID

Table 4.2-20. stationTable Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
elevation	integer	NULL	-999	99999	Station elevation above MSL
ICAOCallSign	char(8)	NULL	N/A	N/A	Station call sign assigned by the International Civil Aviation Organization
latitude	smallfloat	NULL	-90.0000000	90.0000000	Station latitude
longitude	smallfloat	NULL	-180.0000000	180.0000000	Station longitude
wmoID	char(8)	NULL	N/A	N/A	WMO Block Station number

PRELIMINARY

4.2.3.3.9 Upper Air Profile Table

Table Name: UAProfile

Description: Stores information computed from an upper air sounding.

Primary Key: objectID

Foreign Key: objectID

Table 4.2-21. UAProfile Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
convTemp	smallfloat	NULL	-110.0	63.0	Convection temperature
crossIndex	integer	NULL	0	70	Cross totals index
equilLevel	integer	NULL	0	150000	Height at which the temperature of a buoyantly rising parcel again becomes equal to the temperature of the environment
equilTemp	integer	NULL	-110.0	63.0	Temperature of a buoyantly rising parcel as it becomes equal to the temperature of the environment
heightContrail BPO	integer	NULL	0	150000	Height of the bottom of the layer in which contrail formation is possible
heightContrail TPO	integer	NULL	0	150000	Height of the top of the layer in which contrail formation is possible
heightContrail BPRB	integer	NULL	0	150000	Height of the bottom of the layer in which contrail formation is probable

PRELIMINARY**Table 4.2-21. UAProfile Table Structure**

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
heightContrailTPRB	integer	NULL	0	150000	Height of the top of the layer in which contrail formation is probable
kIndex	integer	NULL	0	50	K Index
liftedIndex	integer	NULL	-10	10	Lifted Index
maxConWindGustDir	integer	NULL	0	359	Maximum convective wind gust direction
maxConWindGustSp	smallfloat	NULL	0	300.0	Maximum convective wind gust speed
maxHailSize	smallfloat	NULL	0	20.0	Maximum diameter of hailstones in cm
objectID	datetime year to fraction (5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique record identifier and time stamp when record was inserted or last updated in the table.
precipWatCon	smallfloat	NULL	0	80.00	Precipitable water content
showalterIndex	integer	NULL	-10	10	Showalter Index
sweatIndex	integer	NULL	0	500	Severe Weather Threat (SWEAT) Index
totalIndex	integer	NULL	9	110	Sum of the vertical and cross totals indexes (Total Totals Index)
vertIndex	integer	NULL	0	40	Vertical totals Index
wetBulbZHeight	integer	NULL	0	150000	Height of wet bulb temperature measurement

PRELIMINARY

4.2.3.3.10 Upper Air Profile Sounding Table

Table Name: uaProfileSounding
Description: Stores vertical profile data computed from an upper air sounding.
Primary Key: objectID/heightType/height
Foreign Key: objectID, heightType, height

Table 4.2-22. uaProfileSounding Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
height	smallint	NOT NULL	0	150000	Height of current level
heightType	char	NOT NULL	SURFACE, STANDARD, TROPOPAUSE, MAX WIND, SIGNIFICANT TEMP, SIGNIFICANT WIND, MISSING		Type of level
mUnits	integer	NULL	0	7000	Modified refractivity units
objectID	datetime year to fraction(5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique record identifier and time stamp when record was inserted or last updated in the table.
potential Temp	integer	NULL	-110.0	63.0	Potential temperature at level
richsNumber	integer	NULL	-5	5	Richardson Number
rlType		NULL			
turbulence Intsty	char	NULL	0	9	Turbulence intensity
turbulence Prob	integer	NULL	0	100	Probability of turbulence at level

PRELIMINARY

Table 4.2-22. uaProfileSounding Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
verticalWindShear	smallfloat	NULL	0.0	300.0	Vertical wind shear at this level

PRELIMINARY

4.2.3.3.11 Upper Air Temperature Table

Table Name: UATemperature
Description: Stores
Primary Key: objectID
Foreign Key: objectID

Table 4.2-23. UATemperature Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
heightLowestIce	smallint	NULL	400	50000	Lowest height at which icing occurred.
heightLowestTurb	char	NULL	0, 2, 4, or 6 (see WMO-306 Code Table 3738)		Lowest height where turbulence is occurring.
launchTime	datetime year to minute	NULL	0-01-01 00:00	9999-12-31 23:59	Time at which radiosonde was launched.
objectID	datetime year to fraction(5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique record identifier and time stamp when record was inserted into the table.
sst	smallfloat	NULL	-5.0	45.0	Sea Surface Temperature. (NULL if report is from a fixed land station).

PRELIMINARY

4.2.3.3.12 Upper Air Temperature Sounding Table

Table Name: UATempSounding
Description: Holds information for one level of an upper air temperature sounding.
Primary Key: objectID/heightType/height
Foreign Key: objectID

Table 4.2-24. UATempSounding Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
airTemperature	smallfloat	NOT NULL	-110.0	63.0	Air temperature at the given height.
geoPotentialHgt	smallint	NOT NULL	0	10000	Height value (dependent on height type).
height	smallint	NOT NULL	0	150000	Value of height in meters, hectopascals, or geopotential meters (dependent on heightType)
heightType	char	NOT NULL	0	7, '/'	Type of height.
moisture	smallfloat	NULL	-110.0 (dew pt) 0 (RH)	(dew pt) 100 (RH)	Moisture value (range dependent on moistureType)
moistureType	char	NULL	0	1	Type of moisture represented by the moisture value (i.e., dewpoint temperature or relative humidity).
objectID	datetime year to fraction(5)	NOT NULL	0-01-01 00:00:00.0 0000	9999-12-31 23:59:59.99 999	A unique record identifier that utilizes the time of the object's insert or update to table.
pressure	smallfloat	NULL	.001	1100.000	Air pressure at given height.

PRELIMINARY

Table 4.2-24. UATempSounding Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
windDirection	smallint	NULL	0	359	Wind direction at the given height.
windSpeed	smallfloat	NULL	0.0	300.0	Wind speed at the given height.

PRELIMINARY

4.2.3.3.13 Upper Air Wind Sounding Table

Table Name: UAWindSounding
Description: Holds wind information for a single level of an upper air wind sounding.
Primary Key: objectID/heightType/height
Foreign Key: objectID

Table 4.2-25. UAWindSounding Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
height	integer	NOT NULL	0	150000	Value of height in meters, millibars, or gpm.
heightType	char	NOT NULL	0	7, '/'	Type of height; valid values are SURFACE, STANDARD, TROPOPAUSE, MAX WIND, SIGNIFICANT TEMPERATURE, and SIGNIFICANT WIND, or MISSING VALUE.
objectID	datetime year to fraction(5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique record identifier and time stamp when record was inserted into the table.
pressure	smallint	NULL	.001	1100.000	Air pressure at given height.
windDirection	smallint	NULL	0	359	Wind direction at given height.
windSpeed	smallfloat	NULL	0.0	300.0	Wind speed at given height.

PRELIMINARY

4.2.3.3.14 Upper Air Report Table

Table Name: UpperAirRep
Description: Stores information about an upper air report.
Primary Key: objectID
Foreign Key: objectID

Table 4.2-26. UpperAirRep Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
instrumentation	byte	NULL	N/A	N/A	Instrumentation used to measure data. For UA TEMP: SSTInstrumentation SIRCorrection trackTechStatus radioSondeUsed For UW: windSensorType thermoDynSensor For Both: measuringEquipment
objectID	datetime year to fraction(5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique Identifier for record, also marks time at which record was inserted or updated.
wind1KM AboveMax	smallfloat	NULL	0.0	300.0	Wind speed at one kilometer above maximum reported wind speed.

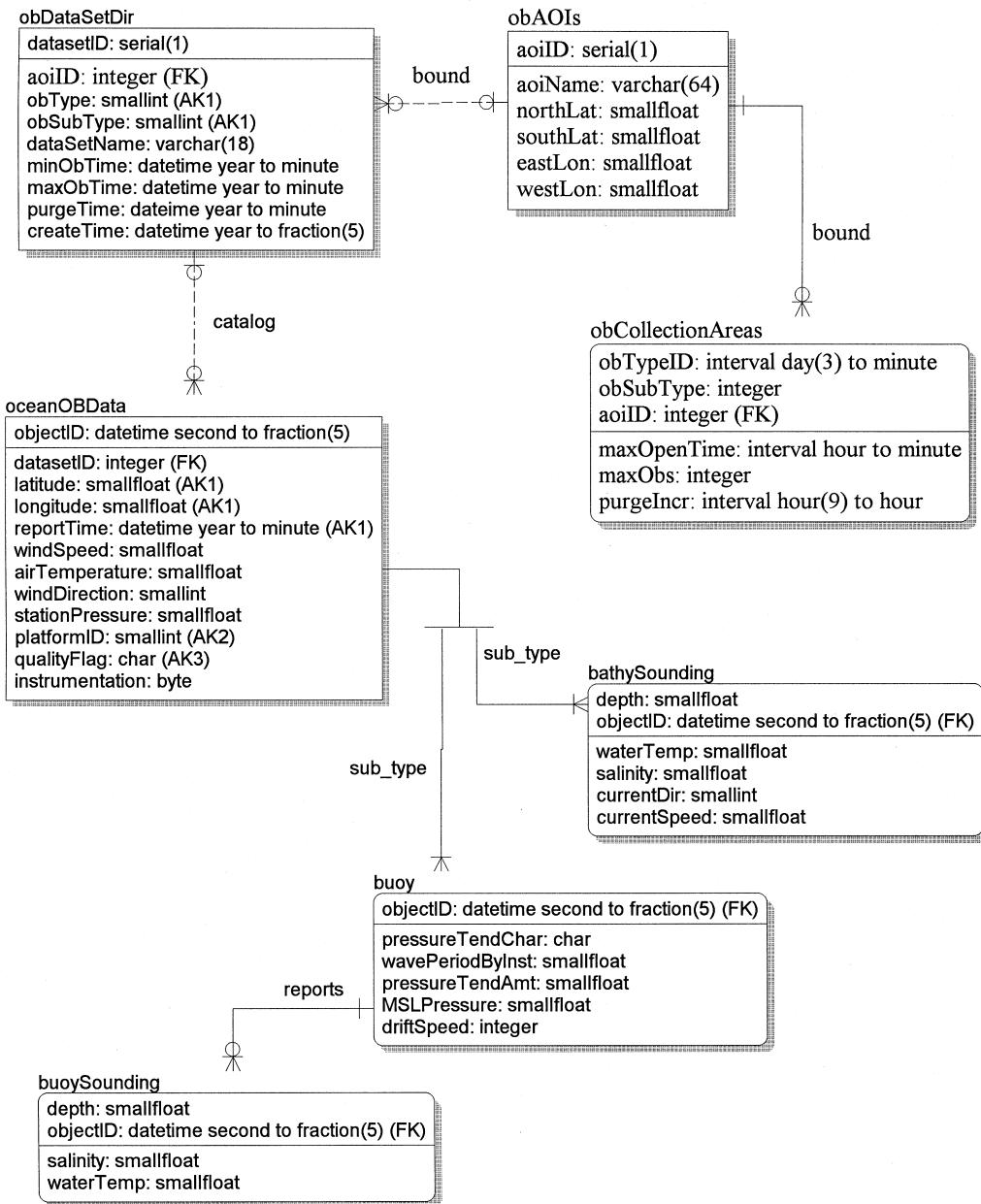
PRELIMINARY

Table 4.2-26. UpperAirRep Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
wind1KM BelowMax	smallfloat	NULL	0.0	300.0	Wind speed at one kilometer below maximum reported wind speed.

PRELIMINARY**4.2.3.4 Physical Level Design for Ocean Observations**

Figure 4.2-10 presents the physical level design for ocean observations as entity-relationship diagrams.

**Figure 4.2-10. Physical Level Design of Storage for Ocean Observations**

The remainder of this section presents the detailed design of the tables used for ocean observations.

PRELIMINARY

4.2.3.4.1 Bathythermograph Sounding Table

Table Name: bathySounding

Description: Stores information for one level of an ocean sounding.

Primary Key: depth/objectID

Foreign Key: objectID

Table 4.2-27. bathySounding Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
currentDir	smallint	NULL	0	35	Direction of current at depth.
currentSpeed	smallfloat	NULL	0.0	300.0	Speed of current at given depth.
depth	smallfloat	NULL	-100.000	12000.000	Depth at which data are reported.
objectID	datetime second to fraction(5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique Identifier for record, also marks time at which record was inserted or updated.
salinity	smallfloat	NULL	0.00	45.00	Salinity of water at depth.
waterTemp	smallfloat	NULL	-2.00	40.00	Temperature of water at depth.

PRELIMINARY

4.2.3.4.2 Buoy Table

Table Name: buoy
Description: Stores sea surface and drift information from a buoy report.
Primary Key: objectID
Foreign Key: objectID

Table 4.2-28. buoy Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
driftSpeed	integer	NULL	0	20	Drift speed of the buoy
MSLPressure	smallfloat	NULL	830.0	1100.0	Mean sea level pressure
objectID	datetime second to fraction(5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique Identifier for record, also marks time at which record was inserted or updated.
pressureTend Amt	smallfloat	NULL	-80.0	80.0	Amount of pressure change
pressureTend Char	char	NULL	0	8	Characteristic of pressure tendency over past 3 hours
wavePeriod ByInst	smallfloat	NULL	0	14	Wave period measured by instrumentation

PRELIMINARY

4.2.3.4.3 Buoy Sounding Table

Table Name: buoySounding
Description: Stores data for a single level of a buoy sounding.
Primary Key: depth/objectID
Foreign Key: objectID

Table 4.2-29. buoySounding Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
depth	smallfloat	NULL	-100	12000	Depth at which data are reported.
objectID	datetime second to fraction(5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique Identifier for record, also marks time at which record was inserted or updated.
salinity	smallfloat	NULL	0.0	45.00	Salinity of water at depth.
waterTemp	smallfloat	NULL	-2.00	40.00	Temperature of water at depth.

PRELIMINARY**4.2.3.4.4 Ocean Observation Data Table****Table Name:** oceanOBData**Description:** Stores data from an ocean observation.**Primary Key:** objectID**Foreign Key:** objectID**Alternate Keys:** latitude/longitude/reportTime, platformID, qualityFlag**Table 4.2-30. oceanOBData Table Structure**

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
airTemperature	smallfloat	NULL	-110.0	63.0	Temperature of air at surface of ocean.
datasetID	integer	NULL	0	232	Unique identifier of a dataset
instrumentation	byte	NULL	N/A	N/A	Codes for instrumentation used to obtain the ocean observation
latitude	smallfloat	NULL	-90.0000000	90.0000000	Latitude of bathy report.
longitude	smallfloat	NULL	-180.0000000	180.0000000	Longitude of bathy.
objectID	datetime second to fraction(5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique Identifier for record, also marks time at which record was inserted or updated.
platformID	smallint	NULL	0	999	Platform identifier
qualityFlag	char	NULL	'0'	'5'	Code for overall quality of observation

PRELIMINARY

Table 4.2-30. oceanOBData Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
reportTime	datetime year to minute	NULL	0001 01-01 00:00	9999 12-31 23:59	Time of report
stationPressure	smallfloat	NULL	450.00	1100.00	Pressure at station level
windDirection	smallint	NULL	001	360	Direction of wind at ocean surface.
windSpeed	smallfloat	NULL	0.00	200.0	Speed of wind at ocean surface.

PRELIMINARY**4.2.3.5 Physical Level Design for METAR, SPECI, and TAF**

Figure 4.2-11 below presents the physical level design for storing METAR, SPECI, and TAF reports as entity-relationship diagrams.

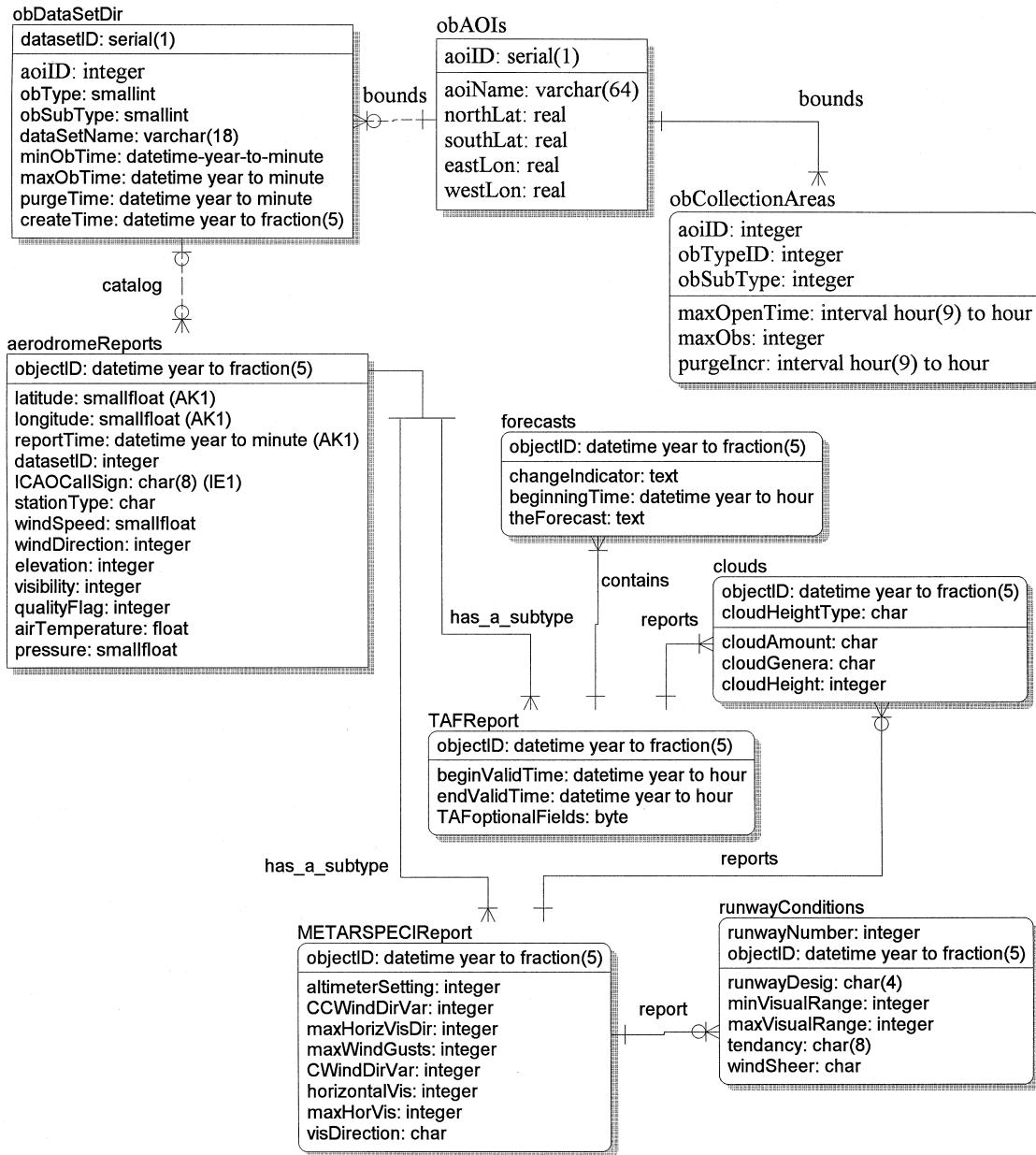


Figure 4.2-11. Physical Level Design for Storage of METAR, SPECI, and TAF Reports

The remainder of this section describes the tables used to store METAR, SPECI, and TAF data.

PRELIMINARY

4.2.3.5.1 Aerodrome Reports Table

Table Name: aerodromeReports
Description: Stores information about the reporting station and station conditions.
Primary Key: objectID

Table 4.2-31. aerodromeReports Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
airTemperature	float	NULL	-110.0	63.0	Air temperature at station
datasetID	integer	NULL	0	232	Unique Identifier of dataset to which observation belongs.
elevation	integer	NULL	-999	90000	Station elevation
ICAOCallSign	char(8)	NULL	N/A	N/A	International Civil Aviation Organization -assigned call sign.
latitude	smallfloat	NULL	-90.0	90.0	Latitude of reporting station.
longitude	smallfloat	NULL	-180.0	180.0	Longitude of reporting station.
objectID	datetime year to fraction(5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique Identifier for record, also marks time at which record was inserted or updated.
pressure	smallfloat	NULL	450.00	1100.00	Station pressure
qualityFlag	integer	NULL			

PRELIMINARY

Table 4.2-31. aerodromeReports Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
reportTime	datetime year to minute	NULL	1997-01-01 00:00	2100-12-31 23:59	Time at which report was issued..
stationType	char	NOT NULL	0	1	Type of station (Manual or Automated)
visibility	integer	NULL	0	10000	Visibility at reporting station
windDirection	integer	NULL	0.0	359	Wind direction at reporting station.
windSpeed	smallfloat	NULL	0.0	200.00	Wind Speed at reporting station.

PRELIMINARY

4.2.3.5.2 Clouds Table

Table Name: cloudData
Description: Contains information concerning amount, height, and type of clouds reported.
Primary Key: objectID/cloudHeightType
Foreign Key: objectID

Table 4.2-32. cloudData Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
cloudAmount	char	NULL	0	9	Amount of Clouds in sky.
cloudGenera	char	NULL	0	9, ‘/’	Genera of cloud.
cloudHeight	integer	NULL	0	99	Altitude of upper surface of clouds whose base is below the level of the station, in hundreds of meters
cloudHeight Type	character	NOT NULL	0	9	Type of cloud: low, middle, or high
objectID	datetime year to fraction(5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique identifier for the object; also its creation time

PRELIMINARY

4.2.3.5.3 Forecasts Table

Table Name: forecasts

Description: Stores forecast data from a terminal aerodrome forecast (TAF).

Primary Key: objectID

Foreign Key: objectID

Table 4.2-33. forecasts Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
beginningTime	datetime year to hour	NOT NULL	1997-01-01 00:00	2100-12-31 23:59	Time at which forecasted condition is expected.
changeIndicator	text	NULL	N/A	N/A	Indicator of weather change FROM, AT, UNTIL, TEMPO, BECMG
objectID	datetime year to fraction(5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique Identifier for record, also marks time at which record was inserted or updated.
theForecast	text	NOT NULL	N/A	N/A	Text of forecast

PRELIMINARY

4.2.3.5.4 METAR/SPECI Report Table

Table Name: METASPECIReport
Description: Stores information from a METAR or SPECI report.
Primary Key: objectID
Foreign Key: objectID

Table 4.2-34. METARSPECIReport Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
altimeterSetting	integer	NULL	600.0	1100.0	Altimeter setting
CCWindDirVar	integer	NULL	0	359	Variation in wind direction in counter-clockwise direction.
CWindDirVar	integer	NULL	0	359	Variation in wind direction in clockwise direction.
horizontalVis	integer	NULL	0	10000	Horizontal visibility at surface.
maxHorizVisDir	integer	NULL	0	359	Direction of maximum horizontal visibility.
maxHorVis	integer	NULL	0	10000	Max horizontal visibility at surface.
maxWindGusts	integer	NULL	0	200	Maximum wind gusts.
objectID	datetime year to fraction(5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique Identifier for record, also marks time at which record was inserted or updated.
visDirection	char	NULL	0	9	Code for direction of visibility restriction

PRELIMINARY

4.2.3.5.5 Runway Conditions Table

Table Name: runwayConditions**Description:** Stores information about runway conditions from METAR, SPECI, and TAF reports.**Primary Key:** objectID/runwayNumber**Foreign Key:** objectID**Table 4.2-35. runwayConditions Table Structure**

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
maxVisualRange	integer	NULL	0	10000	Minimum visual range on runway.
minVisualRange	integer	NULL	0	10000	Minimum visual range on runway.
objectID	datetime year to fraction(5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique Identifier for record, also marks time at which record was inserted or updated.
runwayDesig	char(4)	NULL	N/A	N/A	Runway Designator R,L,C
runwayNumber	integer	NOT NULL	N/A	N/A	Runway information is for
tendency	char(8)	NULL	D, N, or U		Tendency of runway visual range values. U = Increasing, D=Decreasing, N=No distinction.
windShear	char	NULL			Flag indicating existence of wind shear along the take-off path or approach path between one runway level and 500 meters.

PRELIMINARY

4.2.3.5.6 TAF Report Table

Table Name: TAFReport
Description: Stores information from a TAF report.
Primary Key: objectID
Foreign Key: objectID

Table 4.2-36. TAFReport Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
beginValidTime	datetime year to hour	NULL	1997-01-01 00:00	2100-12-31 23:59	Begin valid time of TAF.
endValidTime	datetime year to hour	NULL	1997-01-01 00:00	2100-12-31 23:59	End valid time of TAF.
objectID	datetime year to fraction(5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique Identifier for record, also marks time at which record was inserted or updated.
TAFoptionalFields	byte	NULL	N/A	N/A	Text of TAF optional fields

PRELIMINARY

4.3 Textual Observations Data Segment (MDTXT) Design

4.3.1 MDTXT Conceptual Design

Textual observations are logical groupings of warnings and textually notices, alerts or informational messages that don't fit into WMO formatted messages. Examples of these message would be tropical cyclone warnings, SIGMETS etc. This information will be stored by type of message, valid times of message and are of coverage.

4.3.2 Physical Level Design for Textual Observations and Bulletins Storage

Figure 4.3-1 below shows the physical level design for storage of textual observations and bulletins as entity-relationship diagrams.

PRELIMINARY

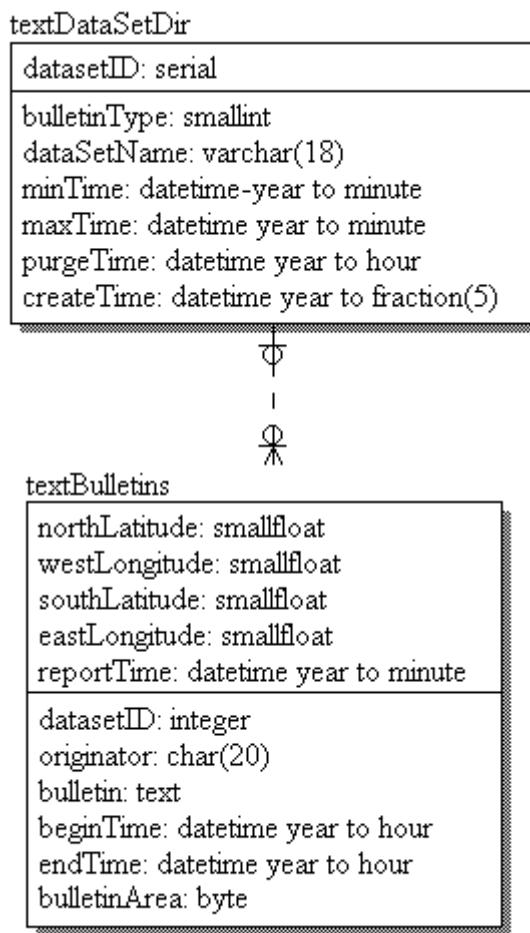


Figure 4.3-1. Physical Level Design for Textual Observations and Bulletins Storage

The remainder of this section describes the tables used to store textual observations and bulletins data.

PRELIMINARY

4.3.2.1 *Text Bulletins Table*

Table Name: textBulletins
Description: Stores information from a single textual observation or bulletin.
Primary Key: eastLongitude/northLatitude/southLatitude/westLongitude/
reportTime
Foreign Key: datasetID

Table 4.3-1. textBulletins Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
beginTime	datetime year to hour	NULL	1997-01-01 00:00	2100-12-31 23:59	Begin valid time of bulletin.
bulletin	text	NOT NULL	N/A	N/A	Text of bulletin
bulletinArea	byte	NULL	N/A	N/A	
datasetID	integer	NULL	0	232	Identification of dataset to which bulletin belongs.
eastLongitude	smallfloat	NOT NULL	-180.0000000	180.0000000	Eastern vector bounding bulletin area.
endTime	datetime year to hour	NOT NULL	1997-01-01 00:00	2100-12-31 23:59	End valid time of bulletin.
northLatitude	smallfloat	NOT NULL	-90.0000000	90.0000000	Northern vector bounding bulletin area.

PRELIMINARY

Table 4.3-1. textBulletins Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
originator	char(20)	NULL	N/A	N/A	Alphanumeric name of bulletin originator
reportTime	datetime year to minute	NOT NULL	1997-01-01 00:00	2100-12-31 23:59	Time of report
southLatitude	smallfloat	NOT NULL	-90.0000000	90.0000000	Southern vector bounding bulletin area.
westLongitude	smallfloat	NOT NULL	-180.0000000	180.0000000	Western vector bounding bulletin area..

PRELIMINARY

4.3.2.2 *Textual Observations/Bulletins Dataset Directory Table*

Table Name: textDataSetDir

Description: Stores information about a textual observation/bulletin dataset.

Primary Key: datasetID

Table 4.3-2. textDataSetDir Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
bulletinType	smallint	NOT NULL			Type of bulletin being stored in the dataset.
createTime	datetime year to fraction(5)	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	time at which dataset was created.
datasetID	serial	NOT NULL	0	232	Unique Identifier of a dataset.
dataSetName	varchar(18)	NOT NULL	N/A	N/A	Alphanumeric name of Ob dataset, also used as name of detail table which stores the grid.
maxTime	datetime year to minute	NULL	0-01-01 00:00	9999-12-31 23:59	Maximum end valid time of a bulletin in the dataset.
minTime	datetime- year to minute	NULL	0-01-01 00:00	9999-12-31 23:59	Minimum begin valid time of a bulletin in the dataset.
purgeTime	datetime year to hour	NOT NULL	0-01-01 00:00	9999-12-31 23:59	Time at which to purge the dataset.

PRELIMINARY

4.4 Image Data Segment (MDIDS) Design

4.4.1 MDIDS Conceptual Design

Image data will support the storage and retrieval of data that are pixel or bit map images. They may be logical grouped by area and time. However there is no requirement that these item be provided. Conceptually, images are a logical group of data that a user wants to store in the database.

4.4.2 MDIDS Logical Level Design

Figure 4.4-1 below shows the logical level design of the MDIDS segment in entity-relationship diagrams.

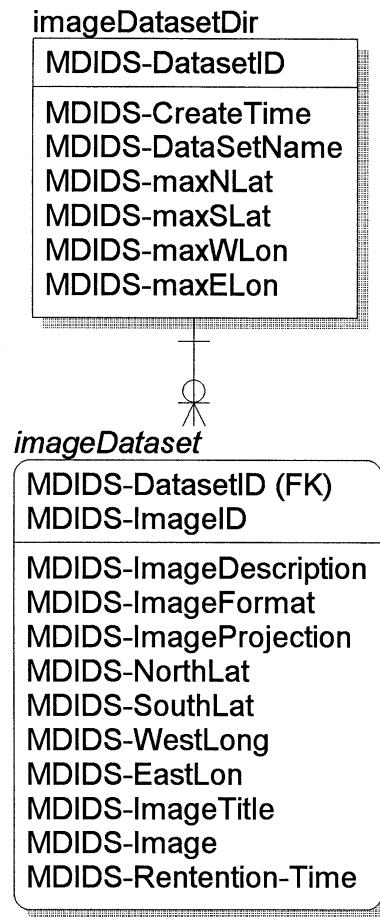


Figure 4.4-1. Logical Level Design for MDIDS

PRELIMINARY

4.4.3 Physical Level Design of MDIDS

Figure 4.4-2 below shows the physical level design of MDIDS in entity-relationship diagrams.

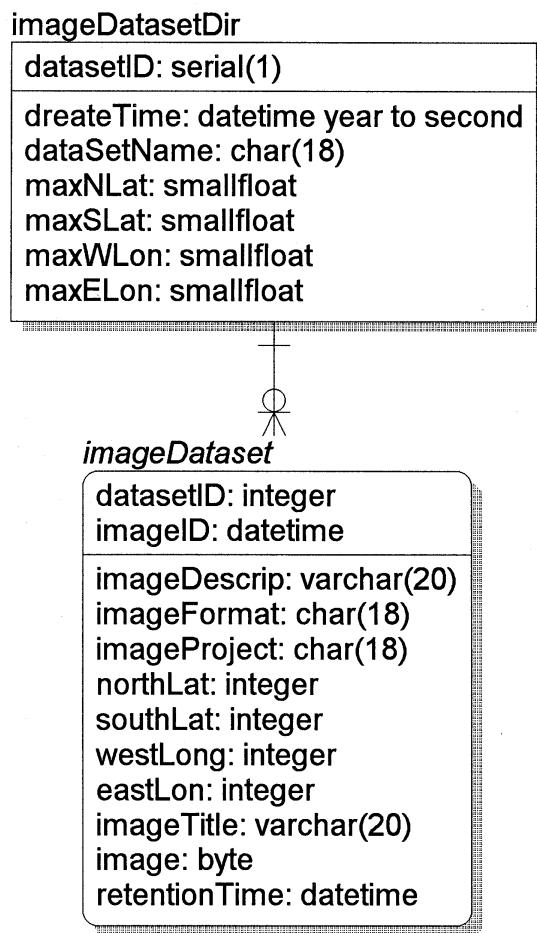


Figure 4.4-2. Physical Level Design of MDIDS

The remainder of this section describes the tables used for storage of image data.

PRELIMINARY

4.4.3.1 *Image Dataset Table*

Table Name: imageDataset
Description: Stores information about an image.
Primary Key: datasetID/ImageID
Foreign Key: datasetID

Table 4.4-1. imageDataset Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
datasetID	integer	NOT NULL	0	232	Unique identifier for dataset
eastLon	integer	NULL	-180.0000000	180.0000000	Easternmost longitude in image
image	byte	NULL	N/A	N/A	Image Data
image Descrip	varchar (20)	NULL	N/A	N/A	Alphanumeric description of the image
imageFormat	char(18)	NULL	N/A	N/A	Image format description
imageID	datetime	NOT NULL	0-01-01 00:00:00.0000 0	9999-12-31 23:59:59.999 99	Unique identifier for the image; also time at which dataset was created.
imageProject	char(18)	NULL	N/A	N/A	Image projection name
imageTitle	varchar (20)	NULL	N/A	N/A	Alphanumeric title of image
northLat	integer	NULL	-90.0000000	90.0000000	Northernmost latitude in image
rentention Time	datetime	NULL	00:00:00	240:00:00	Time to retain the image
southLat	integer	NULL	-90.0000000	90.0000000	Southern-most latitude in image
westLong	integer	NULL	-180.0000000	180.0000000	Westernmost longitude in image

PRELIMINARY

4.4.3.2 *Image Dataset Directory Table*

Table Name: imageDatasetDir

Description: Stores information about an image dataset.

Primary Key: datasetID

Table 4.4-2. imageDatasetDir Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
createTime	datetime year to second	NOT NULL	0-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Time at which dataset was created.
datasetID	serial(1)	NOT NULL	0	232	Unique identifier for dataset
dataSetName	char(18)	NOT NULL	N/A	N/A	Alphanumeric name of dataset
maxELon	smallfloat	NOT NULL	-180.0000000	180.0000000	Eastern-most longitude in dataset
maxNLat	smallfloat	NOT NULL	-90.0000000	90.0000000	Northern-most latitude in dataset
maxSLat	smallfloat	NOT NULL	-90.0000000	90.0000000	Southern-most latitude in dataset
maxWLon	smallfloat	NOT NULL	-180.0000000	180.0000000	Western-most longitude in dataset

PRELIMINARY

4.5 Remotely Sensed Data Segment (MDRSD)

4.5.1 MDRSD Conceptual Level Design

Remotely sensed data is received via the SMQ-11 interface or through BUFR messages and is sensed by satellites. Each satellite will have a sensor or suite of sensors. Each sensor has a channel. Each of the channels sense different environmental phenomenon. The satellite orbits the earth. The information about the satellites orbit (Ephemeris data) is distributed via the MEDS channel on a daily basis. Times and locations of satellite passes can be derived from this information.

The conceptual design of the remotely sensed data is built to model the above facts. However it is assumed that the data stored into this remotely sensed data will be geo-graphically located products rather than raw satellite passes. So actually data will be in a geo-located format assuming that there is a bounding rectangle.

4.5.2 MDRSD Logical Level Design

Figure 4.5-1 presents the logical level design of the MDRSD in entity-relationship diagrams.

PRELIMINARY

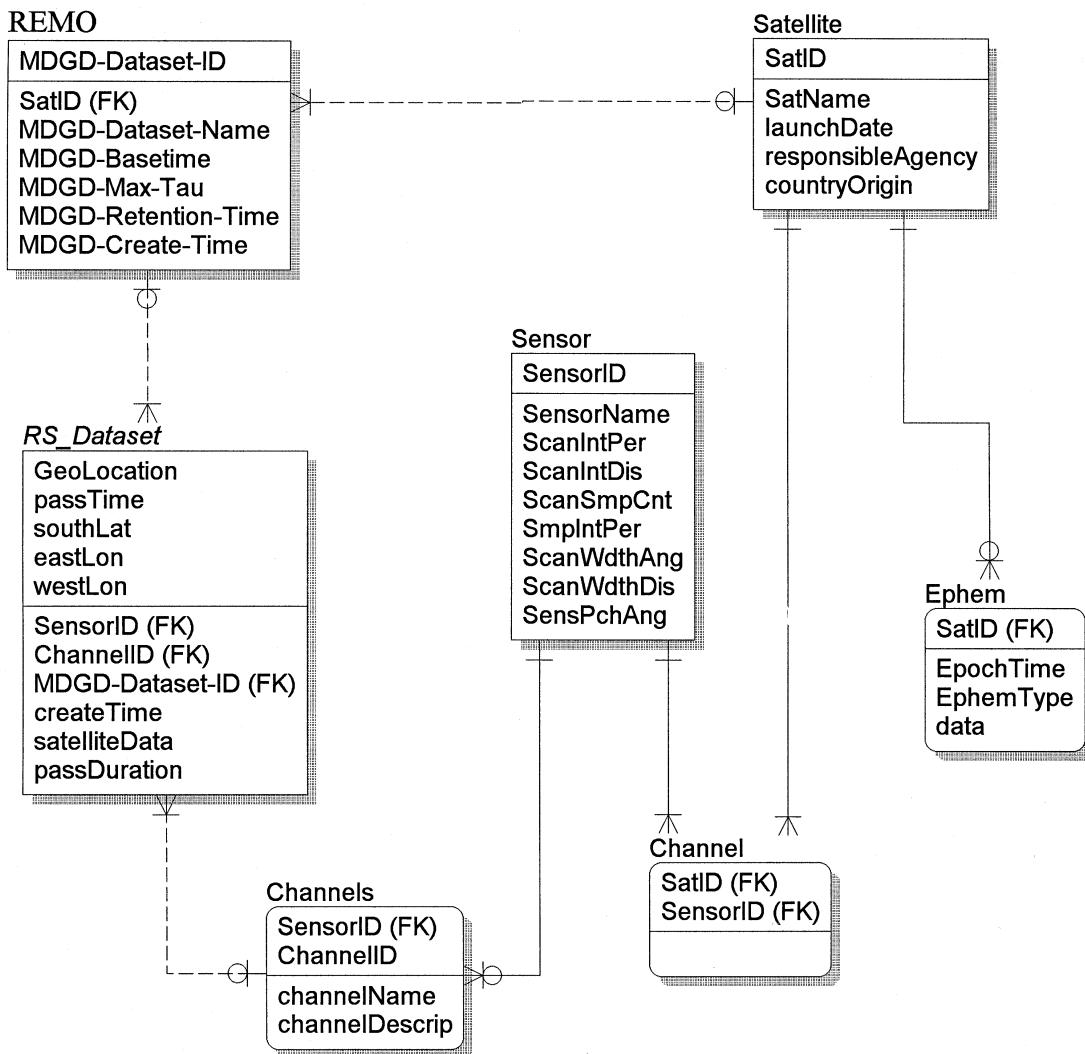


Figure 4.5-1. MDRSD Logical Level Design

PRELIMINARY

4.5.3 MDRSD Physical Level Design

Figure 4.5-2 depicts the physical level design of the MDRSD.

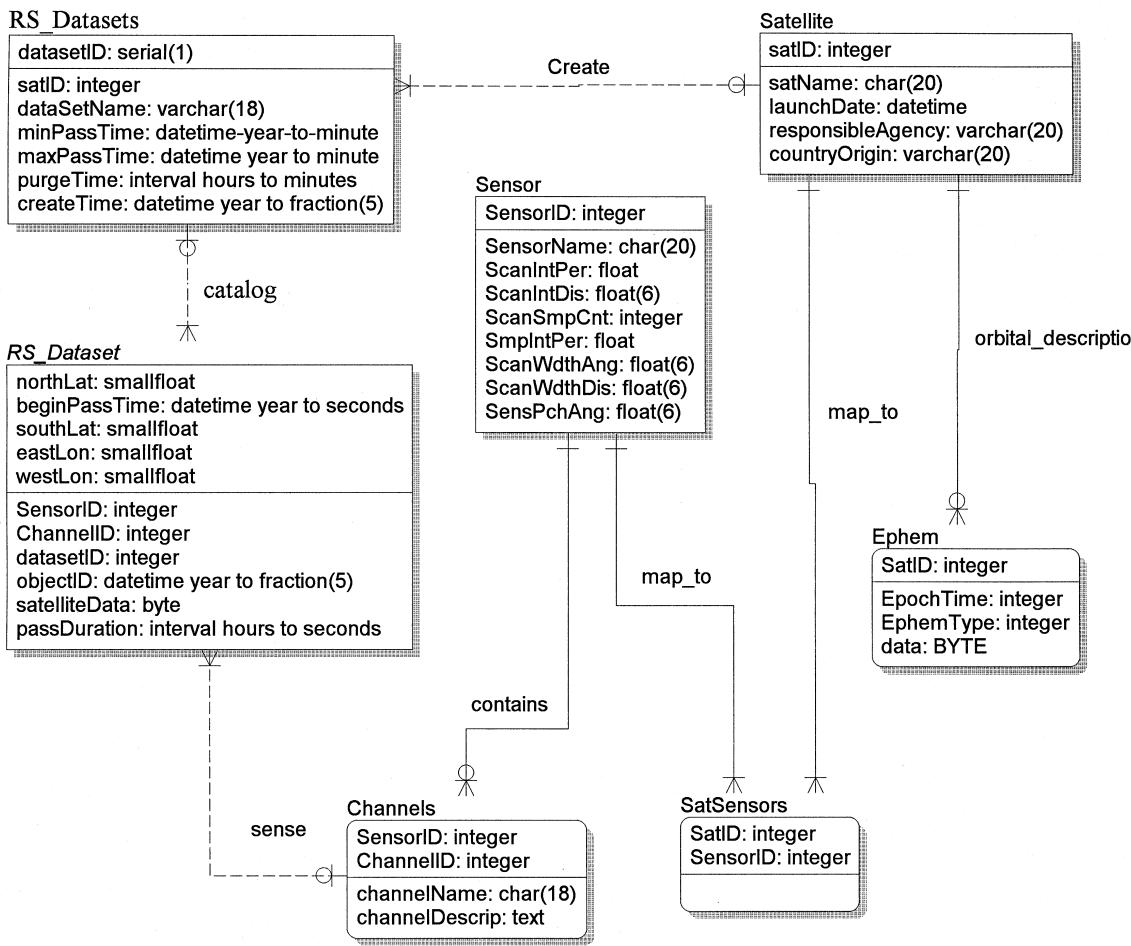


Figure 4.5-2. MDRSD Physical Level Design

The remainder of this section describes the tables that make up the MDRSD physical level.

PRELIMINARY

4.5.3.1 *Channels Table*

Table Name: Channels

Description: Stores channel to sensor cross-reference information.

Primary Key: SensorID/ChannelID

Table 4.5-1. Channels Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
channelDescrip	text	NULL	N/A	N/A	Alphanumeric channel description
ChannelID	integer	NOT NULL	0	232	Unique identifier for channel
channelName	char(18)	NULL	N/A	N/A	Alphanumeric channel name
SensorID	integer	NOT NULL	0	232	ID of sensor to which channel belongs.

PRELIMINARY

4.5.3.2 *Ephemeris Table*

Table Name: Ephem

Description: Stores satellite ephemeris data.

Primary Key: SatID

Table 4.5-2. Ephemeris Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
data	BYTE	NOT NULL	N/A	N/A	Ephemeris data.
EphemType	integer	NOT NULL	0	9	Type of ephemeris (TBUS, C Elements, etc)
EpochTime	integer	NOT NULL			Time of pass.
SatID	integer	NOT NULL	0	232	ID of satellite for which ephemeris applies.

PRELIMINARY

4.5.3.3 *Remotely Sensed Dataset Table*

Table Name: RS_Dataset

Description: Stores a single remotely sensed dataset. This table will be instantiated many times.

Primary Key: northLat/southLat/eastLon/westLon/beginPassTime

Table 4.5-3. RS_Dataset Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
beginPassTime	datetime year to seconds	NOT NULL	0001-01-01 00:00	9999-12-31 23:59	Time at which pass began.
ChannelID	integer	NOT NULL	0	232	Unique identifier for channel
datasetID	integer	NOT NULL	0	232	Identification of dataset to which record belongs.
eastLon	smallfloat	NOT NULL	-180.0	180.0	Easternmost longitude in dataset
northLat	smallfloat	NOT NULL	-90.0	90.0	Northernmost latitude in dataset
objectID	datetime year to fraction(5)	NULL	0001-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Unique identifier of record, time at which record was inserted into the database
passDuration	interval hours to seconds	NULL	00:00:00	23:59:59	Duration of pass.

PRELIMINARY

Table 4.5-3. RS_Dataset Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
satelliteData	byte	NOT NULL	N/A	N/A	Stores actual satellite data in a quad tree.
SensorID	integer	NOT NULL	0	232	Identification of sensor which collected the data.
southLat	smallfloat	NOT NULL	-90.0	90.0	Southernmost latitude in dataset
westLon	smallfloat	NOT NULL	-180.0	180.0	Westernmost longitude in dataset

PRELIMINARY4.5.3.4 *Remotely Sensed Dataset Directory Table***Table Name:** RS_Datasets**Description:** Stores information about remotely sensed datasets.**Primary Key:** datasetID**Table 4.5-4. RS_Datasets Table Structure**

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
createTime	datetime year to fraction(5)	NOT NULL	0001-01-01 00:00:00.00000	9999-12-31 23:59:59.99999	Time at which dataset was created.
datasetID	serial(1)	NOT NULL	0	232	Unique Identifier of a dataset.
dataSetName	varchar(18)	NULL	N/A	N/A	Alpha/Numeric name of Ob dataset, also used as name of detail table which stores the grid.
maxPassTime	datetime year to minute	NOT NULL	1996-01-01 00:00	9999-12-31- 23:59	Maximum pass time of remotely sensed data in the DB.
minPassTime	datetime- year-to- minute	NOT NULL	1996-01-01 00:00	9999-12-31- 23:59	Minimum pass time of remotely sensed record in the DB.
purgeTime	interval hours to minutes	NOT NULL	00:00	240:00	Time at which to purge the dataset.
satID	integer	NULL	0	232	Identifier of satellite from which data was received.

PRELIMINARY

4.5.3.5 *Satellite Table*

Table Name: Satellite

Description: Stores information about a satellite.

Primary Key: SatID

Table 4.5-5. Satellite Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
countryOrigin	varchar(20)	NULL	N/A	N/A	Country which owns the satellite.
launchDate	datetime	NULL	0001-01-01 00:00	9999-12-31 23:59	Date satellite was launched.
responsible Agency	varchar(20)	NULL	N/A	N/A	Agency Responsible for the satellite.
satID	integer	NOT NULL	0	232	ID of satellite
satName	char(20)	NOT NULL	N/A	N/A	Name of Satellite

PRELIMINARY

4.5.3.6 *Satellite Sensors Table*

Table Name: SatSensors

Description: Stores cross-reference between satellites and the sensors they carry.

Primary Key: SatID/SensorID

Table 4.5-6. SatSensors Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
SatID	integer	NOT NULL	0	232	Satellite ID to which sensor belongs.
SensorID	integer	NOT NULL	0	232	Sensor onboard a satellite.

PRELIMINARY

4.5.3.7 *Sensor Table*

Table Name: Sensor

Description: Stores information about a satellite sensor.

Primary Key: SensorID

Table 4.5-7. Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
ScanIntDis	float(6)	NOT NULL			Scan interval distance.
ScanIntPer	float	NOT NULL			Scan interval period.
ScanSmpCnt	integer	NOT NULL			Scan sample count.
ScanWdthAng	float(6)	NOT NULL			Scan width angle.
ScanWdthDis	float(6)	NOT NULL			Scan Width distance.
SensorID	integer	NOT NULL			Identification of the sensor.
SensorName	char(20)	NOT NULL			Name of the sensor.
SensPchAng	float(6)	NOT NULL			Sensor pitch angle.
SmpIntPer	float	NOT NULL			Sample interval period.

PRELIMINARY

4.6 Database Utilities

This section describes utility tables used to perform database administration functions. These tables will be replicated in all segments of the METOC Database.

4.6.1 Create Table Utility Tables

Figure 4.6-1 below shows the physical level design for the tables used to support creation of new tables in the database.

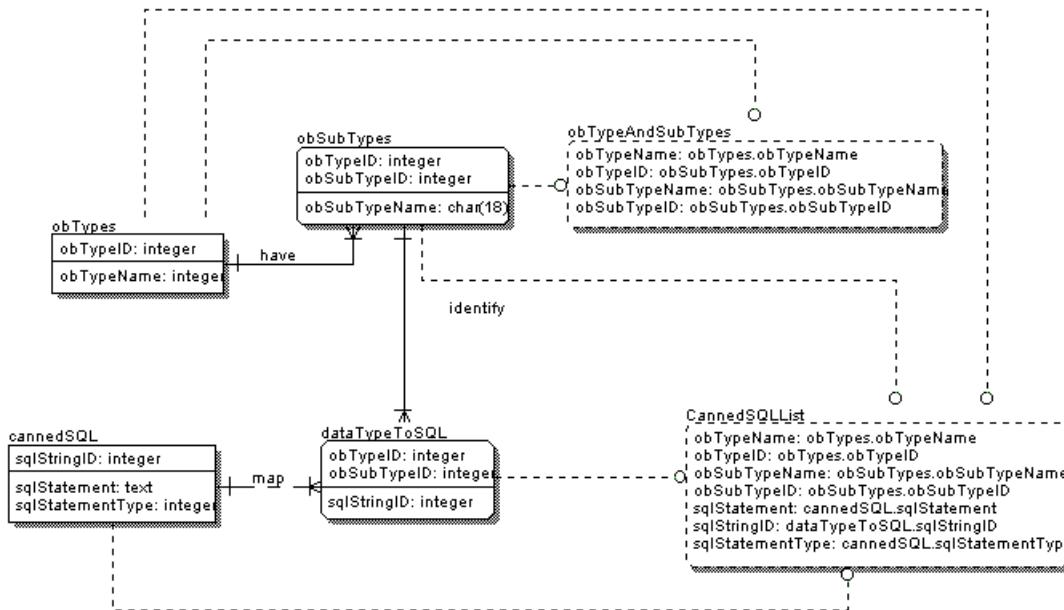


Figure 4.6-1. Physical Design of Create Table Tables

The remainder of this section describes the design of the Create Table tables.

PRELIMINARY

4.6.1.1 Canned SQL Table

Table Name: cannedSQL

Description: Stores SQL statements used to create a table, create an index, grant privilege, or create a stored procedure associated with the tables being created.

Primary Key: sqlStringID

Table 4.6-1. cannedSQL Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
sqlStatement	text	NOT NULL	N/A	N/A	Text of an SQL string to create a table, create an index, or grant privilege, or text for a stored procedure associated with the table(s) being created.
sqlStatementType	integer	NOT NULL	0	232	Type of sql statement in sqlStatement field. Can be create, index, grant or stored procedure.
sqlStringID	integer	NOT NULL	0	232	Unique Identifier of a create string.

PRELIMINARY

4.6.1.2 Data Type to SQL Table

Table Name: dataTypeToSQL
Description: Stores information required to associate an SQL statement with a data type.
Primary Key: obSubTypeID/obTypeID
Foreign Key: obSubTypeID, obTypeID, sqlStringID

Table 4.6-2. dataTypeToSQL Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
obSubTypeID	integer	NOT NULL	0	232	ID of observation subtype
obTypeID	integer	NOT NULL	0	232	ID of observation type
sqlStringID	integer	NOT NULL	0	232	ID of string to be retrieved.

PRELIMINARY

4.6.1.3 *Observation Subtypes Table*

Table Name: obSubTypes
Description: Stores information relating observation subtypes to observation types.
Primary Key: obTypeID/obSubTypeID
Foreign Key: obTypeID

Table 4.6-3. obSubTypes Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
obSubTypeID	integer	NOT NULL	0	232	Unique identifier for observation subtype
obSubTypeName	char(18)	NULL	N/A	N/A	Alphanumeric name of observation subtype
obTypeID	integer	NOT NULL	0	232	Unique identifier for observation type

PRELIMINARY

4.6.1.4 *Observation Types Table*

Table Name: obTypes

Description: Stores information about observation types.

Primary Key: obTypeID

Table 4.6-4. obTypes Table Structure

Table Column Name	Table Column Datatype	Table Column Null Option	Table Column Validation Min	Table Column Validation Max	Table Column Attribute Definition
obTypeID	integer	NOT NULL	0	232	Unique identifier for observation type
obTypeName	char	NULL	N/A	N/A	Alphanumeric name of observation type

PRELIMINARY

5 DETAILED DESIGN OF SOFTWARE UNITS USED FOR DATABASE ACCESS OR MANIPULATION

This section describes the Application Program Interfaces (APIs) provided to access the METOC Database.

5.1 Interfaces for 2-D Gridded Data

5.1.1 Retrieving 2-D Gridded Data From the TEDS Database

Method Name: `ted_GridRetr`

Description: Retrieves 2-D gridded data matching specifications in an input GRIDQUERY structure, returning a linked list of filled GRIDDATA structures. The area, grid geometry, and projection specified are validated. A key is then calculated based on the parameter ID and level ID specified. The reference time to query for is calculated and `ted_GridRetrByKey` is called to retrieve the grid data. This method incorporates virtual products – it will return data which is not contained in the TEDS database but can be created from data that is in the database.

Calling Interface:

```
int ted_GridRetr ( GRIDQUERY GridQuery, LINKEDLIST *pGridDataLL )
```

Inputs: `GridQuery` The GRIDQUERY structure containing the retrieval criteria

Outputs: `pGridDataLL` Pointer to a linked list of GRIDDATA structures containing the retrieved data

`status` 0 if no error, or error code from *tedserr.h*

PRELIMINARY

The GRIDQUERY structure is the key to the retrieval. It provides the specifications used to narrow the retrieval so that only the desired data are retrieved, and so that the data are retrieved in the desired format. The GRIDQUERY structure consists of a GRIDFIELDDESC structure that contains the parameter, time, and area specifications and a GEOMDESC structure that contains the area, model, source, and projection specifications. The individual parameters are:

GRIDFIELDDESC:

lKey (Key for element/level)	This parameter must be specified unless the sParameterID and sLevelID parameters are specified. The key for each parameter ID and level ID is contained in the <i>keys</i> table in the TEDS database and is accessible through the kernel routine <i>krnl_GetKey</i> . There is no advantage, however, to using <i>krnl_GetKey</i> to get the key to enter here, since it requires the parameter ID and level ID as inputs.
lBegRefTime (Earliest reference time)	Epoch time or abstract time indicator specifying the earliest reference (or base) time of the data to be retrieved. The only abstract time indicators valid here are BEGINNING_OF_TIME and TIME_WILDCARD.
lEndRefTime (Latest reference time)	Epoch time or abstract time indicator specifying the latest reference time of the data to be retrieved. This is the time at which the data on which the grid is based were observed. The only abstract time indicators valid here are END_OF_TIME and TIME_WILDCARD. This parameter must be greater than or equal to lBeginRefTime.

PRELIMINARY

lBegValidTime (Earliest valid time)	Epoch time or abstract time indicator specifying the earliest valid time of the data to be retrieved. This is the earliest time at which the grid data requested are valid, and is equal to lBeginRefTime plus the forecast hour. That is, to obtain a 24-hour forecast based on data from lBeginRefTime, lBeginValidTime would be lBeginRefTime + 86400 seconds. The only abstract time indicators valid here are BEGINNING_OF_TIME and TIME_WILDCARD.
lEndValidTime (Latest valid time)	Epoch time or abstract time indicator specifying the latest valid time of the data to be retrieved. This is latest the time at which the grid data requested are valid, and is equal to lEndRefTime plus the forecast hour.
sParameterID (Parameter ID)	TEDS identifier for the parameter to be retrieved. Must be specified if lKey is not specified. Parameter IDs are contained in the <i>parms</i> table of the TEDS database (See Appendix A, Table A-1, for a listing of the contents of this table as of this writing). This table may be accessed through the kernel routines <i>krnl_ParmByName</i> (to get parameter information given the parameter name) and <i>krnl_ParmByID</i> (to get parameter information given the parameter ID). See Appendix B for information on calling these routines and others related to parameters.

PRELIMINARY

sLevelID (Level ID)	TEDS identifier for the level for which data are to be retrieved. Must be specified if lKey is not specified. Level data are contained in the <i>Levels</i> table in the TEDS database, whose current contents are listed in Appendix A, Table A-2. This table is accessible through the kernel routines <i>krnl_LevelByName</i> (to get LevelID and units given the level name), <i>krnl_LevelUnit</i> (to get units and level name based on LevelID), and <i>krnl_LevelID</i> (to get level name based on Level value and units). See Appendix B for information on calling these routines and others dealing with levels.
SzGeomName (Geometry name)	Name of the geometry in which data are to be retrieved (e.g. GLOBAL_73X144). May be wildcarded. This information is contained in the TEDS <i>modelRef</i> table, whose current contents are displayed in Appendix A, Table A-3. The primary access to this table is provided by the kernel routine <i>krnl_GetModelRef</i> . See Appendix B for information on calling this routine.
szParamUnits (Physical units)	Units in which the data are to be retrieved. If this is different than the units in which the data are stored in the database, a conversion is done during the retrieval. Must be specified. The available units are stored in the TEDS <i>unitDesc</i> table, whose current contents are displayed in Appendix A, Table A-4. The primary access to this table is provided by the kernel routine <i>krnl_GetUnitDesc</i> . See Appendix B for information on calling this routine and others dealing with units.

PRELIMINARY

SzModelName (Model name)	Name of the model whose output is to be retrieved (e.g. NOGAPS, OTIS, etc.). May be wildcarded. This information is contained in the TEDS <i>modelRef</i> table, whose current contents are displayed in Appendix A, Table A-3. The primary access to this table is provided by the kernel routine <i>krnl_GetModelRef</i> . See Appendix B for information on calling this routine.
GEOMDESC:	
eStoreDesc (Storage descriptor)	Specifies the order of the data within the output. Must be specified. Valid descriptors are pXinnY, pXinpY, nYinpX, and pYinpX (defined in the file <i>GeomReg.h</i>). See Appendix C for explanations of these descriptors.
eProjection (Projection)	Specifies the projection in which the data are to be returned. Must be specified. At present only spherical projections (RWS_SPHERICAL) are reliably supported.
rdNorthLat (Northernmost latitude)	Northernmost latitude of retrieval area. Must be specified.
rdSouthLat (Southernmost latitude)	Southernmost latitude of retrieval area. Must be specified.
rdEastLong (Easternmost longitude)	Easternmost longitude of retrieval area. Must be specified.
rdWestLong (Westernmost longitude)	Westernmost longitude of retrieval area. Must be specified.

PRELIMINARY

nX_Dim (Grid points in X)	Number of grid points in the X direction (between rdEastLong and rdWestLong) to be included in the output. Must be specified.
nY_Dim (Grid points in Y)	Number of grid points in the Y direction (between rdNorthLat and rdSouthLat) to be included in the output. Must be specified.

PRELIMINARY

5.1.2 Storing 2-D Gridded Data in the TEDS Database

Method Name: ted_GridStore

Description: Stores 2-D gridded data in the TEDS database. The data to be stored are input in a GRIDDATA structure. A pointer to a dataset reference structure for the dataset containing the data is returned.

Calling Interface:

```
int ted_Gridstore (GRIDDATA *pGridData, DATASETREF **ppDatasetRef)
```

Inputs: pGridData Pointer to the GRIDDATA structure containing the data to be stored

Outputs: ppDatasetRef Handle for the DATASETREF structure for the dataset containing the stored grid

status 0 if no error, or error code from *tedserr.h*

An alternative method for storing 2-D gridded data is to use the two routines *ted_Open2D* and *ted_Add2GridDS*, described in the following 2 sections.

PRELIMINARY

5.1.3 Opening a 2-D Grid Dataset

Method Name: ted_Open2d

Description: Opens a dataset containing 2-D gridded data, or creates a new dataset if the one requested does not exist.

Calling Interface:

```
int ted_Open2D ( char *szDSName; char *szDSDir;
                  long *lDSID; long lMode;
                  char *szUModelName;
                  char *szUGeomName )
```

Inputs: szDSName Name of the dataset to be opened.

szDSDir Name of the directory table containing datasets.
Passing a null value indicates that the routine should use the default table, datasetDir. This table will be used by most users.

lDSID Index into the dataset manager table for use in later calls to the dataset manager.

lMode Mode in which to open the dataset:
READ_ONLY: Open for read only
READ_WRITE: Open for read and write
READ_WRITE_CREATE: Open for read and write, will create dataset if it doesn't exist.

szUModelName Name of the grid model associated with the data.

szUGeomName Name of the grid geometry associated with the data.

Outputs: lDSID Index into the dataset manager table for use in later calls to the dataset manager.

Return status 0 if no error, or error code from *tedserr.h*

PRELIMINARY

5.1.4 Adding 2-D Gridded Data to a Dataset

Method Name: ted_Add2GridDS

Description: Adds 2-D gridded data to a dataset.

Calling Interface:

```
int ted_Add2GridDS(long lSID, PDATASETDET pDetails, void *theData)
```

Inputs:	lSID	Dataset identifier.
	Pdetails	Pointer to the dataset detail structure.
	TheData	The data record to be added to the dataset.
Outputs:	Return status	0 if no error, or error code from <i>tedserr.h</i>

PRELIMINARY

5.1.5 Retrieving a Catalog Listing of 2-D Gridded Data

Method Name: ted_GridRetrCat

Description: Retrieves a catalog listing of 2-D gridded data in the TEDS database that matches the criteria set in the input DirQuery structure. This method incorporates virtual products – the catalog listing will include products that are not in the database but can be created by TEDS from information that is in the database.

Calling Interface:

```
int ted_GridRetrCat ( GRIDQUERY GridQuery, long *lNumFound,  
                      LINKEDLIST *pCatList )
```

Inputs: GridQuery GridQuery structure containing search criteria for the data catalog to be retrieved.

Outputs: lNumFound Number of catalog entries found that matched the search criteria

pCatList Pointer to the head of a linked list of GRIDREFERENCE structures for data matching the search criteria

status 0 if no error, or error code from *tedserr.h*

PRELIMINARY

5.1.6 Retrieving 2-D Gridded Data By Reference ID

Method Name: ted_GridRetrRef

Description: Retrieves 2-D gridded data from a specific dataset, based on criteria in a GRIDREFERENCE structure, which contains both GRIDQUERY information identifying the data to be retrieved and a DATASETREF structure identifying the table in which the grid resides. This method incorporates virtual products – it will return data which is not contained in the TEDS database but can be created from data that is in the database.

Calling Interface:

```
int ted_GridRetrRef ( GRIDREFERENCE *pGridReference,  
                      GRIDDATA **ppGridData )
```

Inputs: pGridReference Pointer to the GRIDREFERENCE structure

Outputs: ppGridData Handle to the GRIDDATA structure containing the retrieved data

status 0 if no error, or error code from *tedserr.h*

PRELIMINARY

5.2 Interfaces for Bathythermograph Report (BT) Data

5.2.1 Getting a Catalog of BT Data From the Database

Method Name: ted_GetBTCatalog

Description: Queries the database for BT data matching the criteria in the input CATALOGQUERY structure and returns a linked list of CATALOGDATA structures for data matching the query criteria.

Calling Interface:

```
int ted_GetBTCatalog ( PCATALOGQUERY pCatalogQuery,  
                      int*pNumRetrieved,  
                      PLINKEDLIST pLLCatalogData )
```

Inputs: pCatalogQuery Pointer to a CATALOGQUERY structure containing the search criteria.

Outputs: pNumRetrieved Pointer to number of records found that matched the search criteria.

pLLCatalogData Pointer to a linked list of CATALOGDATA structures containing catalog information for records that matched the search criteria.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.2.2 Retrieving a Specific BT Record From the Database

Method Name: ted_GetBT

Description: Retrieves a specific BT record, given the table name and record ID. The BT data are returned in a BTSTRUCT structure.

Calling Interface:

```
int ted_GetBT      (   double rdRecID, char *szTableName,  
                      PBTSTRUCT pBathy )
```

Inputs: rdRecID Identifier for the BT record to be retrieved.

szTableName Table where the record to be retrieved resides.

Outputs: pBathy Pointer to the BTSTRUCT structure containing the retrieved data.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.2.3 Retrieving Multiple BT Records From the Database

Method Name: ted_GetBTs

Description: Takes as input a CATALOGQUERY structure providing search criteria, and returns a linked list of BTSTRUCT structures containing data from records that matched the input criteria.

Calling Interface:

```
int ted_GetBTs      ( PCATALOGQUERY pCatalogQuery,  
                      int *pNumRetrieved,  
                      PLINKEDLIST pBTStructsLL )
```

Inputs: pCatalogQuery Pointer to a CATALOGQUERY structure containing search criteria.

Outputs: pNumRetrieved Pointer to number of records found that matched the search criteria.

pBTStructsLL Pointer to a linked list of BTSTRUCT structures containing the retrieved data.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.2.4 Deleting a BT Record From the Database

Method Name: ted_DeleteBT

Description: Takes as inputs a table name and a record ID identifying a BT record. Deletes the appropriate record from the table.

Calling Interface:

```
int ted_DeleteBT ( double rdRecID, char *szTableName )
```

Inputs: rdRec Identifier for the BT record to be deleted.

szTableName Table where the record to be deleted resides.

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.2.5 Adding a BT Record to the Database

Method Name: ted_AddBT

Description: Adds the BT data contained in the input BTSTRUCT structure to the database. Returns the table name and record ID of the data.

Calling Interface:

```
int ted_AddBT      ( PBTSTRUCT pBathy, double *pRecID,  
                      char *szTableName )
```

Inputs: pBathy A pointer to a BTSTRUCT structure containing the BT data to be added to the database.

Outputs: pRecID Pointer to the identifier for the BT record.

szTableName Name of the table where the BT data are stored.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.2.6 Adding a BT Record to a Dataset

Method Name: ted_Add2BTDS

Description: Adds the BT data pointed to by pObs to the dataset identified by lDSID.

Calling Interface:

```
int ted_Add2BTDS ( long lDSID, POBS pObs )
```

Inputs: lDSID Identifier for the dataset to which BT data is to be added.

pObs Pointer to the BT data to be added to the dataset

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.2.7 Updating a BT Record

Method Name: ted_UpdateBT

Description: Updates the BT record identified by the given record ID in the specified table with data from the input BTSTRUCT structure.

Calling Interface:

```
int ted_UpdateBT ( PBTSTRUCT pBTStruct, char *szTableName  
double rdRecID )
```

Inputs: pBTStruct Pointer to a BTSTRUCT structure containing BT data used to update the specified record.

szTableName Table where the record to be updated resides.

rdRecID Identifier for the BT record to be updated.

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.3 Interfaces for Ocean Profile (OP) Data

5.3.1 Getting a Catalog of OP Data From the Database

Method Name: `ted_GetOceanProfileCatalog`

Description: Queries the database for OP data matching the criteria in the input CATALOGQUERY structure and returns a linked list of CATALOGDATA structures for data matching the query criteria.

Calling Interface:

```
int ted_GetOceanProfileCatalog( PCATALOGQUERY pCatalogQuery,
                               int *pNumRetrieved,
                               PLLCatalogData )
```

Inputs: `pCatalogQuery` Pointer to a CATALOGQUERY structure containing the search criteria.

Outputs: `pNumRetrieved` Pointer to number of records found that matched the search criteria.

`pLLCatalogData` Pointer to a linked list of CATALOGDATA structures containing catalog information for records that matched the search criteria.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.3.2 Retrieving a Specific OP Record From the Database

Method Name: ted_GetOceanProfile

Description: Retrieves a specific OP record, given the table name and profile ID. The OP data are returned in an OCEANPROFILE structure.

Calling Interface:

```
int ted_GetOceanProfile ( double rdProfileID,  
                         char *szTableName,  
                         POCEANPROFILE pOceanProfile )
```

Inputs: rdRecID Record identifier for the record to be retrieved.

szTableName Table where the record to be retrieved resides.

Outputs: pOceanProfile Pointer to the OCEANPROFILE structure containing the retrieved data.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.3.3 Retrieving Multiple OP Records From the Database

Method Name: ted_GetOceanProfiles

Description: Takes as input a CATALOGQUERY structure providing search criteria, and returns a linked list of OCEANPROFILE structures containing data from records that matched the input criteria.

Calling Interface:

```
int ted_GetOceanProfiles ( PCATALOGQUERY pCatalogQuery,  
                           int *pNumRetrieved,  
                           PLINKEDLIST pOPsLL )
```

Inputs: pCatalogQuery Pointer to a CATALOGQUERY structure containing search criteria.

Outputs: pNumRetrieved Pointer to number of records found that matched the search criteria.

pOPsLL Pointer to a linked list of OCEANPROFILE structures containing the retrieved data.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.3.4 Deleting an OP Record From the Database

Method Name: ted_DeleteOceanProfile

Description: Takes as inputs a table name and a profile ID identifying an OP record. Deletes the appropriate record from the table.

Calling Interface:

```
int ted_DeleteOceanProfile ( double rdProfileID,  
                           char *szTableName )
```

Inputs: rdProfileID Identifier for the OP record to be deleted.

szTableName Table where the record to be deleted resides.

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.3.5 Adding an OP Record to the Database

Method Name: ted_AddOceanProfile

Description: Adds the OP data contained in the input OCEANPROFILE structure to the database. Returns the table name and record ID of the data.

Calling Interface:

```
int ted_AddOceanProfile ( POCEANPROFILE pOceanProfile,  
                         double *pProfileID,  
                         char *szTableName )
```

Inputs: pOceanProfile Pointer to an OCEANPROFILE structure containing the OP data to be added to the database.

Outputs: pProfileID Identifier for the record where the OP data are stored.

szTableName Name of the table where the OP data is stored.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.3.6 Adding an OP Record to a Dataset

Method Name: ted_Add2OPDS

Description: Adds the OP data pointed to by pObs to the dataset identified by lDSID.

Calling Interface:

```
int ted_Add2OPDS ( long lDSID, POBS pObs )
```

Inputs: lDSID Identifier for the dataset to which OP data is to be added.

pObs Pointer to the OP data to be added to the dataset

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.3.7 Updating an OP Record

Method Name: ted_UpdateOceanProfile

Description: Updates the OP record identified by the given record ID in the specified table with data from the input OCEANPROFILE structure.

Calling Interface:

```
int ted_UpdateOceanProfile (      POCEANPROFILE pOceanProfile,  
                           char *szTableName,  
                           double rdProfileID )
```

Inputs: pOceanProfile Pointer to an OCEANPROFILE structure containing data used to update the specified record.

szTableName Table where the record to be updated resides.

rdProfileID Identifier for the OP record to be updated.

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.4 Interfaces For Upper Air Report (UA) Data

5.4.1 Getting a Catalog of UA Data From the Database

Method Name: ted_GetUACatalog

Description: Queries the database for UA data matching the criteria in the input CATALOGQUERY structure and returns a linked list of CATALOGDATA structures for data matching the query criteria.

Calling Interface:

```
int ted_GetUACatalog ( PCATALOGQUERY pCatalogQuery,
                      int *pNumRetrieved,
                      PLINKEDLIST pLLCatalogData )
```

Inputs: pCatalogQuery Pointer to a CATALOGQUERY structure containing the search criteria.

Outputs: pNumRetrieved Pointer to number of records found that matched the search criteria.

 pLLCatalogData Pointer to a linked list of CATALOGDATA structures containing catalog information for records that matched the search criteria.

 Return status 0 if no error, or error status from *tedserr.h*.

PRELIMINARY

5.4.2 Retrieving a Specific UA Record From the Database

Method Name: ted_GetUA

Description: Retrieves a specific UA record, given the table name and record ID. The UA data return in an UASTRUCT structure.

Calling Interface:

```
int ted_GetUA ( double rdRecID, char *szTableName,  
                PUASTRUCT pUpperAir )
```

Inputs: rdRecID Identifier for the UA record to be retrieved.

szTableName Table where the record to be retrieved resides.

Outputs: pUpperAir Pointer to the UASTRUCT structure containing the retrieved data.

Return status 0 if no error, or error status from *tedserr.h*.

PRELIMINARY

5.4.3 Retrieving Multiple UA Records From the Database

Method Name: ted_GetUAs

Description: Takes as input a CATALOGQUERY structure providing search criteria, and returns a linked list of UASTRUCT structures containing data from records that matched the input criteria.

Calling Interface:

```
int ted_GetUAs      ( PCATALOGQUERY pCatalogQuery,  
                      int *pNumRetrieved,  
                      PLINKEDLIST pUAStructsLL )
```

Inputs: pCatalogQuery Pointer to a CATALOGQUERY structure containing search criteria.

Outputs: pNumRetrieved Pointer to number of records found that matched the search criteria.

pUAStructsLL Pointer to a linked list of UASTRUCT structures containing the retrieved data.

Return status 0 if no error, or error status from *tedserr.h*.

PRELIMINARY

5.4.4 Deleting an UA Record From the Database

Method Name: ted_DeleteUA

Description: Takes as inputs a table name and a record ID identifying an UA record. Deletes the record from the table.

Calling Interface:

```
int ted_DeleteUA ( double rdRecID, char *szTableName )
```

Inputs: rdRecID Identifier for the UA record to be deleted.

szTableName Table where the record to be deleted resides.

Outputs: Return status 0 if no error, or error status from *tedserr.h*.

PRELIMINARY

5.4.5 Adding a UA Record to the Database

Method Name: ted_AddUA

Description: Adds the UA data contained in the input UASTRUCT structure to the database. Returns the table name and record ID of the data.

Calling Interface:

```
int ted_AddUA      ( PUASTRUCT pUpperAir,  
                      double *pRecID, char *szTableName )
```

Inputs: pUpperAir Pointer to a UASTRUCT structure containing the UA data to be added to the database.

Outputs: pRecID Identifier for the record where the UA data are stored.

szTableName Name of the table where the UA data are stored.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.4.6 Adding a UA Record to a Dataset

Method Name: ted_Add2UADS

Description: Adds the UA data pointed to by pObs to the dataset identified by lDSID.

Calling Interface:

```
int ted_Add2UADS ( long lDSID, POBS pObs )
```

Inputs: lDSID Identifier for the dataset to which UA data is to be added.

pObs Pointer to the UA data to be added to the dataset

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.4.7 Updating a UA Record

Method Name: ted_UpdateUA

Description: Updates the UA record identified by the given record ID in the specified table with data from the input UASTRUCT structure.

Calling Interface:

```
int ted_UpdateUA ( PUASTRUCT pUAStruct, char *szTableName,  
                   double rdRecID )
```

Inputs: pUAStruct Pointer to a UASTRUCT structure containing data used to update the specified record.

szTableName Table where the record to be updated resides.

rdRecID Identifier for the UA record to be updated.

Outputs: Return status 0 if no error, or error status from *tedserr.h*.

PRELIMINARY

5.5 Interfaces For Upper Air Profile (UP) Data

5.5.1 Getting a Catalog of UP Data From the Database

Method Name: `ted_GetUPCatalog`

Description: Queries the database for UP data matching the criteria in the input CATALOGQUERY structure and returns a linked list of CATALOGDATA structures for data matching the query criteria.

Calling Interface:

```
int ted_GetUPCatalog (   PCATALOGQUERY pCatalogQuery,
                        int *pNumRetrieved,
                        PLINKEDLIST pLLCatalogData )
```

Inputs: `pCatalogQuery` Pointer to a CATALOGQUERY structure containing the search criteria.

Outputs: `pNumRetrieved` Pointer to number of records found that matched the search criteria.

`pLLCatalogData` Pointer to a linked list of CATALOGDATA structures containing catalog information for records that matched the search criteria.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.5.2 Retrieving a Specific UP Record From the Database

Method Name: ted_GetUP

Description: Retrieves a specific UP record, given the table name and record ID. The UP data return in a UPSTRUCT structure.

Calling Interface:

```
int ted_GetUP      ( double rdRecID, char *szTableName,  
                      PUPSTRUCT pUpperAirProfile )
```

Inputs: rdRecID Identifier for the UP record to be retrieved.

TableName Table where the record to be retrieved resides.

Outputs: pUpperAirProfile Pointer to the UPSTRUCT structure containing the retrieved data.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.5.3 Retrieving Multiple UP Records From the Database

Method Name: ted_GetUPs

Description: Takes as input a CATALOGQUERY structure providing search criteria, and returns a linked list of UPSTRUCT structures containing data from records that matched the input criteria.

Calling Interface:

```
int ted_GetUPs      ( PCATALOGQUERY pCatalogQuery,  
                      int *pNumRetrieved,  
                      PLINKEDLIST pUPStructsLL )
```

Inputs: pCatalogQuery Pointer to a CATALOGQUERY structure containing search criteria.

Outputs: pNumRetrieved Pointer to number of records found that matched the search criteria.

pUPStructsLL Pointer to a linked list of UPSTRUCT structures containing the retrieved data.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.5.4 Deleting a UP Record From the Database

Method Name: ted_DeleteUP

Description: Takes as inputs a table name and a record ID identifying an UP record. Deletes the record from the table.

Calling Interface:

```
int ted_DeleteUP ( double rdRecID, char *szTableName )
```

Inputs: rdRecID Identifier for the UP record to be deleted.

szTableName Table where the record to be deleted resides.

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.5.5 Adding an UP Record to the Database

Method Name: ted_AddUP

Description: Adds the UP data contained in the input UPSTRUCT structure to the database. Returns the table name and record ID of the data. Upper air profiles in TEDS can contain calculated as well as observed data and therefor differ from UA reports.

Calling Interface:

```
int ted_AddUP      (    PUPSTRUCT pUPStruct, double *pRecID,  
                      char *szTableName )
```

Inputs: pUPStruct A pointer to a UPSTRUCT structure containing the UP data to be added to the database.

Outputs: pRecID Pointer to the identifier for the UP record.

szTableName Name of the table where the UP data is stored.

Return status 0 if no error, or error status from *tedserr.h*.

PRELIMINARY

5.5.6 Adding a UP Record to a Dataset

Method Name: ted_Add2UPDS

Description: Adds the UP data pointed to by pObs to the dataset identified by lDSID.

Calling Interface:

```
int ted_Add2UPDS ( long lDSID, POBS pObs )
```

Inputs: lDSID Identifier for the dataset to which UP data is to be added.

pObs Pointer to the UP data to be added to the dataset

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.5.7 Updating a UP Record

Method Name: ted_UpdateUP

Description: Updates the UP record identified by the given record ID in the specified table with data from the input UPSTRUCT structure.

Calling Interface:

```
int ted_UpdateUP ( PUPSTRUCT pUPStruct, char *szTableName,  
                   double rdRecID )
```

Inputs: pUPStruct Pointer to a UPSTRUCT structure containing data used to update the specified record.

szTableName Table where the record to be updated resides.

rdRecID Identifier for the UP record to be updated.

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.6 Interfaces For Synoptic Report (SN) Data

5.6.1 Getting a Catalog of SN Data From the Database

Method Name: ted_GetSNCatalog

Description: Queries the database for SN data matching the criteria in the input CATALOGQUERY structure and returns a linked list of CATALOGDATA structures for data matching the query criteria.

Calling Interface:

```
int ted_GetSNCatalog ( PCATALOGQUERY pCatalogQuery,  
                      int *pNumRetrieved,  
                      PLINKEDLIST pLLCatalogData )
```

Inputs: pCatalogQuery Pointer to a CATALOGQUERY structure containing the search criteria.

Outputs: pNumRetrieved Pointer to number of records found that matched the search criteria.

pLLCatalogData Pointer to a linked list of CATALOGDATA structures containing catalog information for records that matched the search criteria.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.6.2 Retrieving a Specific SN Record From the Database

Method Name: ted_GetSN

Description: Retrieves a specific SN record, given the table name and record ID. The SN data are returned in a SYNSTRUCT structure.

Calling Interface:

```
int ted_GetSN      (   double rdRecID, char *szTableName,  
                      PSYNSTRUCT pSynoptic )
```

Inputs: rdRecID Identifier for the SN record to be retrieved.

szTableName Table where the record to be retrieved resides.

Outputs: pSynoptic Pointer to the SYNSTRUCT structure containing the retrieved data.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.6.3 Retrieving Multiple SN Records From the Database

Method Name: ted_GetSNs

Description: Takes as input a CATALOGQUERY structure providing search criteria, and returns a linked list of SYNSTRUCT structures containing data from records that matched the input criteria.

Calling Interface:

```
int ted_GetSNs      ( PCATALOGQUERY pCatalogQuery,
                      int *pNumRetrieved,
                      PLINKEDLIST pSYNStructsLL )
```

Inputs: pCatalogQuery Pointer to a CATALOGQUERY structure containing search criteria.

Outputs: pNumRetrieved Pointer to number of records found that matched the search criteria.

pSYNStructsLL Pointer to a linked list of SYNSTRUCT structures containing the retrieved data.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.6.4 Deleting a SN Record From the Database

Method Name: ted_DeleteSN

Description: Takes as inputs a table name and a record ID identifying a SN record. Deletes the appropriate record from the table.

Calling Interface:

```
int ted_DeleteSN ( double rdRecID, char *szTableName )
```

Inputs: rdRecID Identifier for the SN record to be deleted.

szTableName Table where the record to be deleted resides.

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.6.5 Adding a SN Record to the Database

Method Name: ted_AddSN

Description: Adds the SN data contained in the input SYSNSTRUCT structure to the database. Returns the table name and record ID of the data.

Calling Interface:

```
int ted_AddSN      ( PSYNSTRUCT Synoptic, double *pRecID,  
                      char *szTableName )
```

Inputs: Synoptic A pointer to a SYNSTRUCT structure containing the SN data to be added to the database.

Outputs: pRecID Pointer to the identifier for the SN record.

szTableName Name of the table where the SN data are stored.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.6.6 Adding a SN Record to a Dataset

Method Name: ted_Add2SNDS

Description: Adds the SN data pointed to by pObs to the dataset identified by lDSID.

Calling Interface:

```
int ted_Add2SNDS ( long lDSID, POBS pObs )
```

Inputs: lDSID Identifier for the dataset to which SN data is to be added.

pObs Pointer to the SN data to be added to the dataset

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.6.7 Updating an SN Record

Method Name: ted_UpdateSN

Description: Updates the SN record identified by the given record ID in the specified table with data from the input SYNSTRUCT structure.

Calling Interface:

```
int ted_UpdateSN ( PSYNSTRUCT pSYNStruct,  
                   char *szTableName, double rdRecID )
```

Inputs: pSYNStruct Pointer to a SYNSTRUCT structure containing data used to update the specified record.

 szTableName Table where the record to be updated resides.

 rdRecID Identifier for the SN record to be updated.

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.7

Interfaces For Vertical Sounding (VS) Data

There are 3 types of Vertical Sounding data: TOVS, T1, and T2. When using the CATALOGQUERY structure, the following constants (from tdVS.h) can be used to fill in the “Type” field: TEDS_VS_TOVS, TEDS_VS_T1, TEDS_VS_T2.

5.7.1

Getting a Catalog of VS Data From the Database

Method Name: ted_GetVSCatalog

Description: Queries the database for VS data matching the criteria in the input CATALOGQUERY structure and returns a linked list of CATALOGDATA structures for data matching the query criteria.

Calling Interface:

```
int ted_GetVSCatalog ( PCATALOGQUERY pCatalogQuery,
                      int *pNumRetrieved,
                      PLINKEDLIST pLLCatalogData )
```

Inputs: pCatalogQuery Pointer to a CATALOGQUERY structure containing the search criteria.

Outputs: pNumRetrieved Pointer to number of records found that matched the search criteria.

pLLCatalogData Pointer to a linked list of CATALOGDATA structures containing catalog information for records that matched the search criteria.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.7.2 Retrieving a Specific VS Record From the Database

Method Name: ted_GetVS

Description: Retrieves a specific VS record, given the table name and record ID. The VS data are returned in a VSSTRUCT structure.

Calling Interface:

```
int ted_GetVS      ( double rdRecID, char *szTableName,  
                     PVSSTRUCT pVS )
```

Inputs: rdRecID Identifier for the VS record to be retrieved.

szTableName Table where the record to be retrieved resides.

Outputs: pVS Pointer to the VSSTRUCT structure containing the retrieved data.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.7.3 Retrieving Multiple VS Records From the Database

Method Name: ted_GetVSs

Description: Takes as input a CATALOGQUERY structure providing search criteria, and returns a linked list of VSSTRUCT structures containing data from records that matched the input criteria.

Calling Interface:

```
int ted_GetVSS ( PCATALOGQUERY pCatalogQuery,  
                  int *pNumRetrieved,  
                  PLINKEDLIST pVSLL )
```

Inputs: pCatalogQuery Pointer to a CATALOGQUERY structure containing search criteria.

Outputs: pNumRetrieved Pointer to number of records found that matched the search criteria.

pVSLL Pointer to a linked list of VSSTRUCT structures containing the retrieved data.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.7.4 Deleting a VS Record From the Database

Method Name: ted_DeleteVS

Description: Takes as inputs a table name and a record ID identifying a VS record. Deletes the appropriate record from the table.

Calling Interface:

```
int ted_DeleteVS ( double rdRecID, char *szTableName )
```

Inputs: rdRecID Identifier for the VS record to be deleted.

szTableName Table where the record to be deleted resides.

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.7.5 Adding a VS Record to the Database

Method Name: ted_AddVS

Description: Adds the VS data contained in the input VSSTRUCT structure to the database. Returns the table name and record ID of the data.

Calling Interface:

```
int ted_AddSV ( PVSSSTRUCT pVS,  
                 double *pRecID, char *szTableName )
```

Inputs: pVS A pointer to a VSSTRUCT structure containing the VS data to be added to the database.

Outputs: pRecID Pointer to the identifier for the VS record.

SzTableName Name of the table where the VS data are stored.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.7.6 Adding a VS Record to a Dataset

Method Name: ted_Add2VSDS

Description: Adds the VS data pointed to by pObs to the dataset identified by lDSID.

Calling Interface:

```
int ted_Add2VSDS ( long lDSID, POBS pObs )
```

Inputs: lDSID Identifier for the dataset to which VS data is to be added.

pObs Pointer to the VS data to be added to the dataset

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.7.7 Updating a VS Record

Method Name: ted_UpdateVS

Description: Updates the VS record identified by the given record ID in the specified table with data from the input VSSTRUCT structure.

Calling Interface:

```
int ted_UpdateVS ( PVSSTRUCT pVSStruct,  
                   char *szTableName, double rdRecID )
```

Inputs: pVSStruct Pointer to a VSSTRUCT structure containing data used to update the specified record.

 szTableName Table where the record to be updated resides.

 rdRecID Identifier for the VS record to be updated.

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.8 Interfaces for AIRCRAFT (AC) Data

5.8.1 Getting a Catalog of AC Data From the Database

Method Name: ted_GetACCatalog

Description: Queries the database for AC data matching the criteria in the input CATALOGQUERY structure and returns a linked list of CATALOGDATA structures for data matching the query criteria.

Calling Interface:

```
int ted_GetACCatalog ( PCATALOGQUERY pCatalogQuery,  
                      int *PNumRetrieved,  
                      PLINKEDLIST pLLCatalogData )
```

Inputs: pCatalogQuery Pointer to a CATALOGQUERY structure containing the search criteria.

Outputs: pNumRetrieved Pointer to number of records found that matched the search criteria.

pLLCatalogData Pointer to a linked list of CATALOGDATA structures containing catalog information for records that matched the search criteria.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.8.2 Retrieving a Specific AC Record From the Database

Method Name: ted_GetAC

Description: Retrieves a specific AC record, given the table name and record ID. The VS data are returned in a PACSTRUCT structure.

Calling Interface:

```
int ted_GetAC      ( double rdRecID, char *szTableName,  
                      PACSTRUCT pAircraft )
```

Inputs: rdRecID Identifier for the AC record to be retrieved.

szTableName Table where the record to be retrieved resides.

Outputs: pAircraft Pointer to the PACSTRUCT structure containing the retrieved data.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.8.3 Retrieving Multiple AC Records From the Database

Method Name: ted_GetACs

Description: Takes as input a CATALOGQUERY structure providing search criteria, and returns a linked list of PACSTRUCT structures containing data from records that matched the input criteria.

Calling Interface:

```
int ted_GetACs ( PCATALOGQUERY pCatalogQuery,  
                 int *pNumRetrieved,  
                 PLINKEDLIST pACStructsLL )
```

Inputs: pCatalogQuery Pointer to a CATALOGQUERY structure containing search criteria.

Outputs: pNumRetrieved Pointer to number of records found that matched the search criteria.

pACStructsLL Pointer to a linked list of ACSTRUCT structures containing the retrieved data.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.8.4 Deleting an AC Record From the Database

Method Name: ted_DeleteAC

Description: Takes as inputs a table name and a record ID identifying an AC record. Deletes the appropriate record from the table.

Calling Interface:

```
int ted_DeleteAC ( double rdRecID, char *szTableName )
```

Inputs: rdRecID Identifier for the AC record to be deleted.

szTableName Table where the record to be deleted resides.

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.8.5 Adding an AC Record to the Database

Method Name: ted_AddAC

Description: Adds the AC data contained in the input PACSTRUCT structure to the database. Returns the table name and record ID of the data.

Calling Interface:

```
int ted_AddAC      ( PACSTRUCT pAircraft,  
                      double *pRecID, char *szTableName )
```

Inputs: pAircraft A pointer to a ACSTRUCT structure containing the AC data to be added to the database.

Outputs: pRecID Pointer to the identifier for the AC record.

szTableName Name of the table where the AC data are stored.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.8.6 Adding an AC Record to a Dataset

Method Name: ted_Add2ACDS

Description: Adds the AC data pointed to by pObs to the dataset identified by lDSID.

Calling Interface:

```
int ted_Add2ACDS ( long lDSID, POBS pObs )
```

Inputs: lDSID Identifier for the dataset to which AC data is to be added.

pObs Pointer to the AC data to be added to the dataset

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.8.7 Updating an AC Record

Method Name: ted_UpdateAC

Description: Updates the AC record identified by the given record ID in the specified table with data from the input ACSTRUCT structure.

Calling Interface:

```
int ted_UpdateAC ( PACSTRUCT pACStruct,  
                   char *szTableName, double rdRecID )
```

Inputs: pACStruct Pointer to a ACSTRUCT structure containing data used to update the specified record.

szTableName Table where the record to be updated resides.

rdRecID Identifier for the AC record to be updated.

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.9 Interfaces For SSM/I Observation Data

5.9.1 Getting a Catalog of SSM/I Data from the Database

Method Name: ted_GetSSMICatalog

Description: Queries the database for SSM/I data matching the criteria in the input CATALOGQUERY structure and returns a linked list of CATALOGDATA structures for data matching the query criteria.

Calling Interface:

```
int ted_GetSSMICatalog ( PCATALOGQUERY pCatalogQuery,  
                          int *pNumRetrieved,  
                          PLINKEDLIST pLLCatalogData )
```

Inputs: pCatalogQuery Pointer to a CATALOGQUERY structure containing the search criteria.

Outputs: pNumRetrieved Pointer to number of records found that matched the search criteria.

 pLLCatalogData Pointer to a linked list of CATALOGDATA structures containing catalog information for records that matched the search criteria.

 Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.9.2 Retrieving a Specific SSM/I Record From the Database

Method Name: ted_GetSSMI

Description: Retrieves a specific SSM/I record, given the table name and record ID. The SSMI data are returned in an SSMISTRUCT structure.

Calling Interface:

```
int ted_GetSSMI ( double rdRecID,  
                  char *szTableName,  
                  PSSMISTRUCT pSSMI )
```

Inputs: tfRecID Identifier for the SSM/I record to be retrieved.

szTableName Table where the record to be retrieved resides.

Outputs: pSSMI Pointer to the SSMISTRUCT structure containing the retrieved data.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.9.3 Retrieving Multiple SSM/I Records From the Database

Method Name: ted_GetSSMIs

Description: Takes as input a CATALOGQUERY structure providing search criteria, and returns a linked list of SSMISTRUCT structures containing data from records that matched the input criteria.

Calling Interface:

```
int ted_GetSSMIs ( PCATALOGQUERY pCatalogQuery,  
                    int *pNumRetrieved,  
                    PLINKEDLIST pSSMILL )
```

Inputs: pCatalogQuery Pointer to a CATALOGQUERY structure containing search criteria.

Outputs: pNumRetrieved Number of records found that matched the search criteria.

pSSMILL Pointer to a linked list of SSMISTRUCT structures containing the retrieved data.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.9.4 Deleting an SSM/I Record From the Database

Method Name: ted_DeleteSSMI

Description: Takes as inputs a table name and a record ID identifying an SSM/I record. Deletes the appropriate record from the table.

Calling Interface:

```
int ted_DeleteSSMI ( double rdRecID,  
                      char *szTableName )
```

Inputs: rdRecID Identifier for the SSM/I record to be deleted.

szTableName Table where the record to be deleted resides.

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.9.5 Adding an SSM/I Record to the Database

Method Name: ted_AddSSMI

Description: Adds the SSM/I data contained in the input PSSMISTRUCT structure to the database. Returns the table name and record ID of the data.

Calling Interface:

```
int ted_AddSSMI ( PSSMISTRUCT pSSMI, double *pRecID,  
                  char *szTableName )
```

Inputs: pSSMIStruct A pointer to an PSSMISTRUCT structure containing the SSM/I data to be added to the database.

Outputs: pRecID Pointer to the identifier for the SSM/I record.

szTableName Name of the table where the SSM/I data are stored.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.9.6 Adding an SSM/I Record to a Dataset

Method Name: ted_Add2SSMIDS

Description: Adds the SSM/I data pointed to by pObs to the dataset identified by lDSID.

Calling Interface:

```
int ted_Add2SSMIDS ( long lDSID, POBS pObs )
```

Inputs: lDSID Identifier for the dataset to which SSM/I data is to be added.

pObs Pointer to the SSM/I data to be added to the dataset

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.9.7 Updating an SSM/I Record

Method Name: ted_UpdateSSMI

Description: Updates the SSM/I record identified by the given record ID in the specified table with data from the input SSMISTRUCT structure..

Calling Interface:

```
int ted_UpdateSSMI ( PSSMISTRUCT pSSMIData,  
                      char *szTableName,  
                      double rdRecID )
```

Inputs: pSSMIData Pointer to the SSMISTRUCT structure containing data used to update the specified record.

szTableName Table where the record to be updated resides.

rdRecID Identifier for the SSM/I record to be updated.

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.10 Interfaces for Textual Observations and Bulletins

5.10.1 Getting a Catalog of TXT Data from the Database

Method Name: ted_GetTXTCatalog

Description: Queries the database for TXT data matching the criteria in the input CATALOGQUERY structure and returns a linked list of CATALOGDATA structures for data matching the query criteria. Note that it is possible to query for a specific textual observation/bulletin subtype by setting the appropriate value from *tdTXT.h* in the *Type* member of the CATALOGQUERY structure.

Calling Interface:

```
int ted_GetTXTCatalog ( PCATALOGQUERY pCatalogQuery,
                        int *pNumRetrieved,
                        PLINKEDLIST pLLCatalogData )
```

Inputs: pCatalogQuery Pointer to a CATALOGQUERY structure containing the search criteria.

Outputs: pNumRetrieved Pointer to number of records found that matched the search criteria.

pLLCatalogData Pointer to a linked list of CATALOGDATA structures containing catalog information for records that matched the search criteria.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.10.2 Retrieving a Specific TXT Record From the Database

Method Name: ted_GetTXT

Description: Retrieves a specific TXT record, given the table name and record ID. The TXT data are returned in a TXTSTRUCT structure.

Calling Interface:

```
int ted_GetTXT ( double rdRecID,  
                 char *szTableName,  
                 PTXTSTRUCT pTXT )
```

Inputs: rdRecID Identifier for the TXT record to be retrieved.

szTableName Table where the record to be retrieved resides.

Outputs: pTXT Pointer to the TXTSTRUCT structure containing the retrieved data.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.10.3 Retrieving Multiple TXT Records From the Database

Method Name: ted_GetTXTs

Description: Takes as input a CATALOGQUERY structure providing search criteria, and returns a linked list of TXTSTRUCT structures containing data from records that matched the input criteria. Note that it is possible to query for a specific textual observation/bulletin subtype by setting the appropriate value from *tdTXT.h* in the *Type* member of the CATALOGQUERY structure.

Calling Interface:

```
int ted_GetTXTs ( PCATALOGQUERY pCatalogQuery,  
                  int *pNumRetrieved,  
                  PLINKEDLIST pTXTLL )
```

Inputs: pCatalogQuery Pointer to a CATALOGQUERY structure containing search criteria.

Outputs: pNumRetrieved Pointer to number of records found that matched the search criteria.

pTXTLL Pointer to a linked list of TXTSTRUCT structures containing the retrieved data.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.10.4 Deleting a TXT Record From the Database

Method Name: ted_DeleteTXT

Description: Takes as inputs a table name and a record ID identifying a TXT record. Deletes the appropriate record from the table.

Calling Interface:

```
int ted_DeleteTXT ( double rdRecID, char *szTableName )
```

Inputs: rdRecID Identifier for the TXT record to be deleted.

szTableName Table where the record to be deleted resides.

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.10.5 Adding a TXT Record to the Database

Method Name: ted_AddTXT

Description: Adds the TXT data contained in the input TXTSTRUCT structure to the database. Returns the table name and record ID of the data.

Calling Interface:

```
int ted_AddSSMI ( PTXTSTRUCT pTXT,  
                   double *pRecID, char *szTableName )
```

Inputs: pTXT A pointer to an TXTSTRUCT structure containing the TXT data to be added to the database.

Outputs: pRecID Pointer to the identifier for the TXT record.

szTableName Name of the table where the TXT data are stored.

Return status 0 of no error, or error code from *tedserr.h*.

PRELIMINARY

5.10.6 Adding a TXT Record to a Dataset

Method Name: ted_Add2TXTDS

Description: Adds the TXT data pointed to by pObs to the dataset identified by lDSID.

Calling Interface:

```
int ted_Add2TXTDS ( long lDSID, POBS pObs )
```

Inputs: lDSID Identifier for the dataset to which TXT data is to be added.

pObs Pointer to the TXT data to be added to the dataset

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.10.7 Updating a TXT Record

Method Name: ted_UpdateTXT

Description: Updates the TXT record identified by the given record ID in the specified table with data from the input TXTSTRUCT structure..

Calling Interface:

```
int ted_UpdateTXT ( PTXTSTRUCT pTXTStruct,  
                    char *szTableName,  
                    double rdRecID )
```

Inputs: pTXTStruct Pointer to the TXTSTRUCT structure containing data used to update the specified record.

szTableName Table where the record to be updated resides.

rdRecID Identifier for the SSM/I record to be updated.

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.11 Interfaces for TRACK WIND Data

5.11.1 Getting a Catalog of TRACKWIND Data from the Database

Method Name: ted_GetTRACKWINDSCatalog

Description: Queries the database for TRACKWINDS data matching the criteria in the input POINT_QUERY structure and returns a linked list of PPOINT_CATALOG structures for data matching the query criteria.

Calling Interface:

```
int ted_GetTrackWindsCatalog( (  
    PPOINT_QUERY pCatalogQuery,  
    int *pNumRetrieved,  
    PLINKEDLIST pTrackWindsLL )
```

Inputs: pCatalogQuery Pointer to a POINT_QUERY structure containing the search criteria.

Outputs: pNumRetrieved Pointer to number of records found that matched the search criteria.

pTrackWindsLL Pointer to a linked list of POINT_CATALOG structures containing catalog information for records that matched the search criteria.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.11.2 Retrieving a Specific TRACKWIND Record From the Database

Method Name: ted_GetTrackWind

Description: Retrieves a specific TRACKWINDS record, given the table name and record ID. The TRACKWINDS data are returned in a TRACKWINDS structure.

Calling Interface:

```
int ted_GetTrackWind ( double rdRecID, char *szTableName,  
                      PTRACKWINDS pTW )
```

Inputs: rdRecID Identifier for the TRACKWINDS record to be retrieved.

SzTableName Table where the record to be retrieved resides.

Outputs: pTW Pointer to the TRACKWINDS structure containing the retrieved data.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.11.3 Retrieving Multiple TRACKWIND Records From the Database

Method Name: ted_GetTrackWinds

Description: Takes as input a POINT_QUERY structure providing search criteria, and returns a linked list of TRACKWINDS structures containing data from records that matched the input criteria.

Calling Interface:

```
int ted_GetTrackWinds ( PPOINT_QUERY pCatalogQuery,  
                        int *pNumRetrieved,  
                        PLINKEDLIST pTrackWindsLL )
```

Inputs: pCatalogQuery Pointer to a POINT_QUERY structure containing search criteria.

Outputs: pNumRetrieved Number of records found that matched the search criteria.

pTrackWindsLL Pointer to a linked list of TRACKWINDS structures containing the retrieved data.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.11.4 Deleting a TRACKWIND Record From the Database

Method Name: ted_DeleteTrackWinds

Description: Takes as inputs a table name and a record ID identifying a TRACKWIND record. Deletes the appropriate record from the table.

Calling Interface:

```
int ted_DeleteTRACKWINDS (double rdRecID, char *szTableName )
```

Inputs: rdRecID Identifier for the TRACKWIND record to be deleted.

szTableName Table where the record to be deleted resides.

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.11.5 TRACKWIND Ingest Interfaces

The following routines are used to ingest track winds data.

Method Name: dsm_OpenDataset

Description: Routine to open up a collection of records.

Calling Interface:

```
int dsm_OpenDataset ( char *szDSName,  
                      char *szDatasetDirectory,  
                      long *plDSID, int nDatatype,  
                      int nSubType, long lMode )
```

Inputs: szDSName: User defined name of a dataset to store a set of track winds data. Can be up to 18 characters in length. Convention is to use ‘tw’ as the prefix concatenated with a date time: tw9707161000.

szDatasetDirectory: Set to NULL; Field is for future use.

nDatatype: Type of data to be stored in the database. Set to TRACK_WINDS which is defined in \$TEDS/INCLUDE/dataTypes.h

nSubType: Type of track winds to store predefined values are:
CLOUD_TRACK_WINDS and
VAPOR_TRACK_WINDS defined in \$TEDS/INCLUDE/dataTypes.h.

lMode: Mode in which to open table. Use RW_CREATE

Outputs: plDSID Numeric ID of dataset that must be provided to the Add function and close functions.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

Method Name: ted_Add2TrackWindsDS(long lDSID, PTWDATASETDET pTW)

Description: Adds a record to the dataset identified by lDSID. lDSID is obtained from a call to dsm_OpenDataset.

Calling Interface:

```
ted_Add2TrackWindsDS ( long lDSID, PTWDATASETDET pTW)
```

Inputs: lDSID: Dataset identifier provided in the dsm_OpenDataset call.

TWData: Address of a track wind record to ingest.

Function Return: 0 For success, or an error defined in \$TEDS/INCLUDE/tedserr.h

Method Name: dsm_CloseDataset(long lDSID)

Description: Closes a dataset that was opened using dsm_OpenDataset for ingest. The dataset may be reopened by making another call to dsm_OpenDataset.

PRELIMINARY

5.11.6 Updating a TRACKWIND Record

Method Name: ted_UpdateTrackWinds

Description: Updates the TRACKWINDS record identified by the given record ID in the specified table with data from the input TRACKWINDS structure..

Calling Interface:

```
int ted_UpdateTrackWinds ( PTRACKWINDS pTRACKWINDSData,  
                           char *szTableName,  
                           double rdRecID )
```

Inputs: szTableName Table where the record to be updated resides.

rdRecID Identifier for the TRACKWIND record to be updated.

Outputs: pTRACKWINDSData Pointer to the TRACKWINDS structure containing data used to update the specified record.

Function Return Value: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.12 Interfaces For Scatterometry Data

5.12.1 Getting a Catalog of SCATTER Data from the Database

Method Name: ted_GetScatterCatalog

Description: Queries the database for SCATTER data matching the criteria in the input POINT_QUERY structure and returns a linked list of POINT_CATALOG structures for data matching the query criteria.

Calling Interface:

```
int ted_GetScatterCatalog (     PPOINT_QUERY pCatalogQuery,
                           int *pNumRetrieved,
                           PLINKEDLIST pScatterLL )
```

Inputs: pCatalogQuery Pointer to a POINT_QUERY structure containing the search criteria.

Outputs: pNumRetrieved Pointer to number of records found that matched the search criteria.

 pScatterLL Pointer to a linked list of POINT_CATALOG structures containing catalog information for records that matched the search criteria.

 Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.12.2 Retrieving a Specific SCATTER Record From the Database

Method Name: ted_GetScatter

Description: Retrieves a specific SCATTER record, given the table name and record ID. The SCATTER data are returned in a SCATTER structure.

Calling Interface:

```
int ted_GetScatter ( double rdRecID, char *szTableName,  
PSCATTER pScat )
```

Inputs: rdRecID Identifier for the SCATTER record to be retrieved.

szTableName Table where the record to be retrieved resides.

Outputs: pScat Pointer to the SCATTER structure containing the retrieved data.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.12.3 Retrieving Multiple Scatterometry Records From the Database

Method Name: ted_GetScatters

Description: Takes as input a POINT_QUERY structure providing search criteria, and returns a linked list of SCATTER structures containing data from records that matched the input criteria.

Calling Interface:

```
int ted_GetScatters ( PPOINT_QUERY pCatalogQuery,  
                      int *pNumRetrieved,  
                      PLINKEDLIST pScatterLL )
```

Inputs: pCatalogQuery Pointer to a POINT_QUERY structure containing search criteria.

Outputs: pNumRetrieved Number of records found that matched the search criteria.

pScatterLL Pointer to a linked list of SCATTER structures containing the retrieved data.

Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.12.4 Deleting a Scatterometry Record From the Database

Method Name: ted_DeleteScatter

Description: Takes as inputs a table name and a record ID identifying a SCATTER record. Deletes the appropriate record from the table.

Calling Interface:

```
int ted_DeleteScatter ( double rdRecID, char *szTableName )
```

Inputs: rdRecID Identifier for the SCATTER record to be deleted.

szTableName Table where the record to be deleted resides.

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.12.5 Ingesting Scatterometry Data

The Following routines are used to ingest scatterometry data. See the example program tGetScat and its source file tGetScat.c for an example on how to use these APIs.

Method Name: dsm_OpenDataset

Description: Routine to open up a collection of records.

Calling Interface:

```
int dsm_OpenDataset ( char *szDSName,
                      char *szDatasetDirectory,
                      long *plDSID, int nDatatype,
                      int nSubType, long lMode )
```

Inputs: szDSName: User defined name of a dataset to store a set of scatterometry data. Can be up to 18 characters in length. Convention is to use ‘sc’ as the prefix concatenated with a date time: sc9707161000.

szDatasetDirectory: Set to NULL; Field is for future use.

nDatatype: Type of data to be stored in the database. Set to SCATTEROMETRY macro, which is defined in \$TEDS/INCLUDE/dataTypes.h

nSubType: Not Used and should be set to zero.

lMode: Mode in which to open table. Use RW_CREATE

Outputs: plDSID: Numeric ID of dataset that must be provided to the Add function and close functions.

PRELIMINARY

Method Name: ted_Add2ScatterDS

Description: Adds a record to the dataset identified by lDSID. lDSID is obtained from a call to dsm_OpenDataset.

Calling Interface:

```
ted_Add2ScatterDS      (   long lDSID; PSCATTER pSC );
```

Inputs: lDSID: Dataset id provided in the dsm_OpenDataset call.

pSC: Address of a SCATTER record to ingest.

Function Return: 0 For success, or an error defined in \$TEDS/INCLUDE/tedserr.h

Method Name: dsm_CloseDataset(long lDSID)

Description: Closes out a dataset that was opened using dsm_OpenDataest for ingest. The dataset may be reopened by making another call to dsm_OpenDataset.

PRELIMINARY

5.12.6 Updating a Scatterometry Record

Method Name: ted_UpdateScatter

Description: Updates the scatterometry record identified by the given record ID in the specified table with data from the input SCATTERSTRUCT structure.

Calling Interface:

```
int ted_UpdateScatter ( PSCATTER pSCATTERData,  
                        char *szTableName,  
                        double rdRecID )
```

Inputs: pSCATTERData Pointer to the SCATTER structure containing data used to update the specified record.

szTableName: Table where the record to be updated resides.

rdRecID Identifier for the SCATTER record to be updated.

Outputs: Return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.13 Interfaces for Image Data

5.13.1 Retrieving TIFF Imagery

Method Name: ted_TIFFRetr

Description: Given a TEDQUERYSAT structure, queries the database and returns a linked list of TEDSAT structures that match the query criteria.

Calling Interface:

```
int ted_TIFFRetr ( TEDQUERYSAT *pTEDQuerySat,  
                   LINKEDLIST *pTEDSatLL)
```

Inputs: pTEDQuerySat Pointer to satellite data query structure

Outputs: pTEDSatLL Pointer to linked list of TEDSAT structures containing data that match the query criteria.

return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

5.13.2 Storing TIFF Imagery in the TEDS Database

Method Name: ted_TIFFStore

Description: Writes a TIFF image and associated information into the TEDS database.

Calling Interface:

```
int ted_TIFFStore ( TEDQUERYSAT *pTEDQuerySat, long lSize,  
                    void *pTIFFBuffer )
```

Inputs: pTEDQuerySat Pointer to satellite query structure containing information associated with the image to be stored.

lSize The size in bytes of the image data.

pTIFFBuffer Pointer to the image data.

Outputs: return status 0 if no error, or error code from *tedserr.h*.

PRELIMINARY

6

REQUIREMENTS TRACEABILITY

This section will provide requirements traceability to the METOC DB SRS.

PRELIMINARY

7 NOTES

7.1 Glossary of acronyms

API	Application Program Interface
DBDD	Database Design Description
DID	Data Item Description
METOC	Meteorology and Oceanography
RDBMS	Relational Database Management System
SPAWAR	Space and Naval Warfare Systems Command
TEDS	Tactical Environmental Data System
TESS	Tactical Environmental Support System